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

ВИКОРИСТАННЯ APACHE SOLR ДЛЯ РЕАЛІЗАЦІЇ ПОВНОТЕКСТОВОГО ПОШУКУ У КОМЕРЦІЙНИХ СИСТЕМАХ

Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки" - 122

Керівник курсової роботи
Яремко С.А.

_____
(підпис)
"____" _____ 2022 р.

Виконав студент КН-1 МП
Кривонос А.А.
"____" _____ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
канд. фіз.-мат. наук, доцент
_____С. С. Гороховський
(підпис)
„____”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Кривоносу Андрію Анатолійовичу факультету інформатики 1-го курсу магістерської програми

ТЕМА: Використання Apache Solr для реалізації повнотекстового пошуку у комерційних системах

Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Анотація
Вступ
Розділ 1. Аналіз предметної області
Розділ 2. Огляд інструментів розробки
Розділ 3. Реалізація системи
Висновки
Список використаної літератури
Додатки

Дата видачі “___” _____ 2022 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Використання Apache Solr для реалізації повнотекстового пошуку у комерційних системах

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	18.10.2021	
2.	Огляд технічної літератури за темою роботи	04.04.2022	
3.	Дослідження існуючих аналогічних програмних рішень	09.04.2022	
4.	Написання теоретичної частини роботи	09.05.2022	
5.	Реалізація ПЗ	23.05.2022	
6.	Написання практичної частини роботи	30.05.2022	
7.	Виправлення помилок та фінальні зміни у текстовій частині	08.06.2022	
8.	Перевірка на плагіат	09.06.2022	
9.	Захист роботи	13.06.2022	

Студент Кривонос Андрій Анатолійович

Керівник Яремко Соломія Андріївна

Зміст

Анотація

У цій роботі досліджено можливості використання Apache Solr у комерційних системах для реалізації повнотекстового пошуку. В процесі дослідження розглянуто основний функціонал та принципи Solr, а також проведено порівняльний аналіз з альтернативною системою ElasticSearch.

Результатом роботи є реалізація практичної частини з використанням Apache Solr та веб-фреймворком FastAPI. Розроблена система виконує функції пошуку по каталогах музичних композицій і призначена для використання стримінговими сервісами.

Вступ

З розвитком технологій та появою великої кількості різноманітних сервісів та систем виникає потреба у організації навігації по величезним обсягах даних, якими ці системи оперують. Звичайний інформаційний пошук засобами СКБД не завжди задовольняє всі потреби розробників програмного забезпечення, зокрема необхідність у швидкості та гнучкості, тому існує велика кількість різноманітних альтернативних рішень. До таких відносяться спеціалізовані пошукові системи та бібліотеки, такі як ElasticSearch, Sphinx Search, Apache Solr, Lunrjs, Apache Nutch, Typesense та інші.

У даній роботі дослідження сфери повнотекстового пошуку зосереджено на пошуковій платформі Apache Solr на прикладі організації пошуку у стримінгових музичних сервісах. Музичні сервіси сьогодні це одна з невід'ємних складових життя сучасної людини. Кількість музичних композицій у Spotify сягає 82 мільйонів, а YouTube Music - більше 80 мільйонів [1, 2], тому зручна навігація по каталогах є однією з основних цілей розробників цих платформ.

Слід зазначити, що у стримінгових музичних сервісах здебільшого роботу по підбору пісень, які подобаються користувачу, виконують рекомендаційні системи за допомогою нейронних мереж. Тим не менш важливість "ручного" пошуку у пісенних каталогах є доволі важливим функціоналом, без якого неможливо уявити жодну сучасну стримінгову платформу.

Метою даної курсової роботи є дослідити можливості пошукової системи Apache Solr, а також розробити на її основі практичний

застосунок для повнотекстового пошуку у каталогах музичних пісень. Основними вимогами до розробки є імплементація API для спрощення інтеграції з іншими сервісами, а також можливість пошуку як по основним даним (назвам пісень, виконавців, альбомам тощо), так і по текстам пісень.

У першому розділі роботи описано загальні відомості про досліджувану сферу, а саме про повнотекстовий пошук, бібліотеку Apache Lucene та пошукові платформи, які засновані на ній, серед яких і Apache Solr.

Другий розділ містить більш технічну інформацію про технології, які використовувались для розробки практичної частини.

Про саму розробку, її архітектуру, проблеми та їх вирішення за допомогою інструментарію Apache Solr розповідається у заключному третьому розділі курсової роботи.

В кінці основної частини підбиті підсумки та зроблені висновки щодо використання Apache Solr для організації пошуку у комерційних системах.

Розділ 1. Аналіз предметної області

1.1 Повнотекстовий пошук

Перед тим як оглядати Apache Solr дамо визначення повнотекстовому пошуку.

Повнотекстовий пошук – це пошук у повнотекстових базах даних. На відміну від іншого типу – бібліографічного – повнотекстові БД містять в собі набори повних текстових документів (книжок, статей, газет, окремих абзаців тощо).

У повнотекстовому пошуку фігурує поняття токену. Токен – це послідовність символів, які розглядаються як семантична одиниця для індексації та пошуку [3]. Інколи токен ототожнюється з поняттям слова, хоча це не є коректним. Багато пошукових систем розглядають як токени частини слів, цілі фрази та знаки пунктуації [4].

Загалом повнотекстовий пошук є доволі важливою функцією, тож такий функціонал підтримується на рівні багатьох сучасних СКБД. Так наприклад Postgresql нативно підтримує базові можливості для такого типу пошуку: токенізація, ігнорування стоп-слів, стемінг, ранжування тощо [5]. Документно-орієнтована NoSQL СКБД MongoDB також підтримує подібний функціонал.

Загалом для організації навігації по відносно невеликих обсягах даних або за відсутності потреби у високій швидкості пошуку засоби СКБД цілком можуть бути достатніми. У інших же випадках варто подивитися у сторону спеціалізованих рішень.

1.2 Бібліотека Apache Lucene

Так однією з найбільш поширених бібліотек для повнотекстового пошуку є Apache Lucene. Як зазначено на головній сторінці офіційного сайту проєкту, команда розробки Lucene займається розвитком та підтримкою ядра пошукової бібліотеки, а також інтеграцію для неї на мові Python [6].

Проєкт був запущений у 1997 році та названий на честь дружини засновника. Як і більшість проєктів, які розглядаються у цій роботі, Apache Lucene написаний на мові Java, яка дозволяє створювати кросплатформові рішення.

Особливістю Apache Lucene є те, що це бібліотека, що дозволяє інтегрувати її у різні системи, в яких необхідно реалізувати пошук. Саме тому проєкт знайшов своє застосування у інтернет-магазинах, а також для створення рекомендаційних систем. Lucene підтримує можливість так званого нечіткого пошуку (fuzzy search), різноманітні форми запитів, ранжування, сортування. Також до переваг можна віднести гарну масштабованість та оптимізованість.

На базі цієї бібліотеки засновано безліч пошукових платформ, серед яких одні з найбільш використовуваних на сьогоднішній день – Apache Solr та ElasticSearch, а також Apache Nutch, Swiftype і DocFetcher.

В наступних розділах буде детальніше описано Apache Solr, оглянуто Elasticsearch, а також проведено поверхневий порівняльний аналіз цих технологій.

1.3 Загальні відомості про Apache Solr

Розробка Apache Solr розпочалась у 2004 році з внутрішнього проєкту компанії CNET [7], яка займається підтримкою порталу, де публікуються статті, подкасти та відео на тему технологій та побутової електроніки. У 2010 Apache Solr був поєднаний з Apache Lucene, після чого вони почали розроблятись спільно некомерційною компанією Apache Software Foundation та спільнотою.

Apache Solr - це open-source пошукова платформа. Серед основних можливостей Solr є повнотекстовий та фасетний пошук, індексування у реальному часі, підтримка кластеризації та масштабування, та зручна інтеграція з базами даних.

Сама платформа Apache Solr працює як окремий сервер, і реалізована на Java, при цьому існує велика кількість клієнтів на різних мовах програмування: Python, Ruby, JavaScript (Node.js), Go, C#, R, C++, Rust, PHP [8]. Крім того, інтерфейс Solr побудований за підходом REST, що дозволяє взаємодіяти з нею через HTTP-запити, використовуючи будь-яку мову програмування без посередництва у вигляді клієнтів.

Слід зазначити, що Solr має можливість налаштування на формат відповідей, який буде зручним для кожного конкретного випадку. Так, це може бути класичний JSON, XML, CSV, або навіть формати структур даних таких мов як Python, Ruby чи PHP. Формат відповіді обирається для кожного запиту окремо, що дозволяє гнучко підходити до розробки застосунків на складних проєктах, де використовуються декілька сервісів, які написані на різних мовах програмування.

Apache Solr має два режими роботи:

- Standalone
- SolrCloud

Режим Standalone призначений для роботи на одному комп’ютері та підходить для тестувальних цілей та реалізації простих додатків. SolrCloud фактично є розширенням і дозволяє розгорнути платформу на кластері.

Кластерний режим надає наступні можливості:

- Реплікація індексу
- Відмовостійкість
- Балансування навантаження
- Розподілені запити

Для координації кластеру використовується інший open-source проєкт – Apache ZooKeeper.

В рамках цієї роботи буде розглядатися лише Standalone режим. Архітектуру Apache Solr в цьому режимі описано у розділі 2.

1.4 Короткий огляд ElasticSearch

ElasticSearch – це розподілений пошуковий сервер, реалізований на основі бібліотеки Apache Lucene [9]. Є наступником проєкту Compass, який розроблявся з 2004 по 2009. У 2010 була оприлюднена перша версія ElasticSearch. Як і Apache Solr, ця платформа є open-source рішенням.

До основних можливостей ElasticSearch відносять:

- Повнотекстовий пошук
- Розподілений пошук
- Підтримка клієнтів на багатьох мовах програмування
- Широкі можливості налаштувань через REST API
- Інтеграції з різними системами та синхронізація з базами даних
- Можливості з налаштування безпеки

1.5 Порівняння Apache Solr та ElasticSearch

Як було згадано раніше, проєкти Apache Solr та ElasticSearch є конкурентами у сфері пошукових платформ. Обидві системи розроблені на основі однієї бібліотеки Apache Lucene, але при цьому мають значні відмінності у реалізації та підходах [10].

Перша відмінність полягає у конфігурації. ElasticSearch налаштовується виключно засобами REST API за допомогою JSON запитів. Apache Solr же можна налаштувати як через файли конфігурації, так і через REST API.

ElasticSearch та Apache Solr мають широкі можливості для кластеризації. При цьому ElasticSearch більш гнучкий в цьому аспекті та дозволяє змінювати конфігурацію кластеру прямо під час обробки запитів через виклики API. Solr є більш статичним і вимагає великої кількості кроків для зміни конфігурації кластеру та не має можливості робити це “на льоту”.

Обидві платформи можуть приймати дані для індексації з великої кількості різних джерел: CSV, XML, JSON файли та з різних БД. ElasticSearch також дозволяє інтегруватись з AWS SQS, Amazon, Redis, Rabitmq та git.

В пошукових аспектах ElasticSearch оптимізований під виконання великої кількості операцій фільтрування та групування. В свою чергу Solr орієнтований на збереження текстів та пошуку по ним.

Загалом можна зробити висновок, що ElasticSearch є більш сучасною системою з більшою кількістю функціональних можливостей та заточений під швидке групування та фільтрацію, тоді як Apache Solr більш стара система, хоча і підтримує велику кількість різних можливостей, а також добре підходить для пошуку по текстовим полям.

Розділ 2. Огляд інструментів розробки

2.1 Apache Solr

Apache Solr може застосовуватись в проєкті як пошукове ядро, і як сховище даних. При цьому існують способи інтеграції Solr з різними БД, що дозволяє застосовувати інструменти СКБД, наприклад, для обчислень, і водночас мати доступ до швидкого пошуку серед текстових даних.

Для початку розглянемо основні компоненти платформи Apache Solr.

2.1.1 Архітектура

Концептуально Apache Solr складається з кількох основних частин. Як згадувалося раніше, основою платформи є бібліотека Apache Lucene. Далі над нею надбудовуються компоненти для обробки запитів (Request Handler), обробки відповідей (Response Writer), а також для оновлення індексу (Update Handler). На рисунку 2.1 зображена дуже спрощений загальний вигляд архітектури Apache Solr.

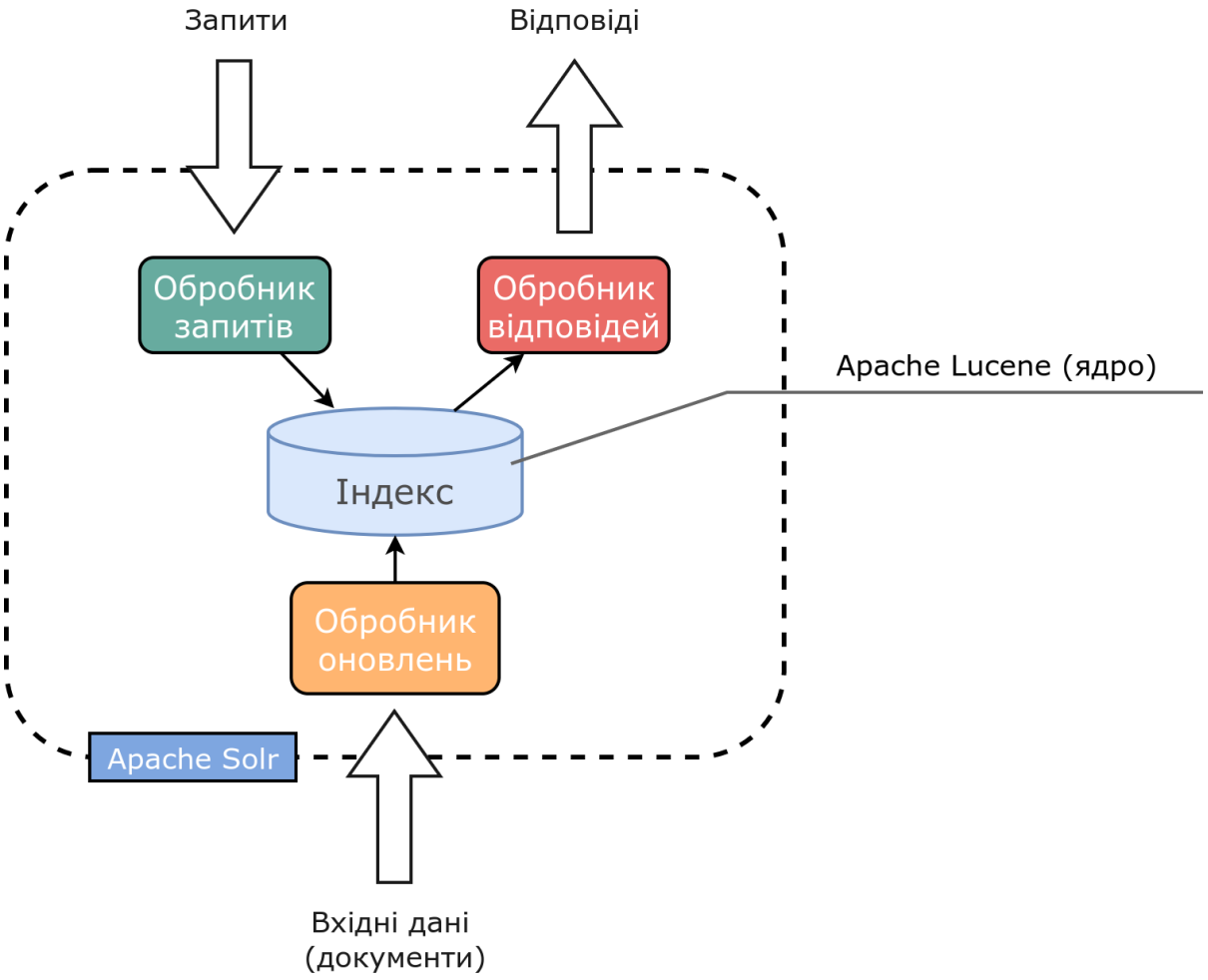

Рисунок 2.1 – Спрощена архітектектура Apache Solr

Індекс поділений на окремі елементи, які в Solr називаються ядрами (core). Кожне ядро містить свої власні налаштування.

2.1.2 Поля і типи

Apache Solr оперує таким поняттям як документ (document). Документ – основна частина інформації про сутності, яка складається з

атрибутів – полів (fields). Поля описують якісь конкретні параметри сутності та мають певний тип – тип поля (field type).

Загалом роботу Apache Solr можна поділити на два етапи:

- Етап індексації (index)
- Етап запитів (query)

Перед індексацією зазвичай створюється схема (про них розказано у наступному підпункті), яка описує відношення полів та їх типів. Вже під час індексації обробник оновлень аналізує вхідні дані і приводить їх до зазначеної у схемі форми.

Під отримання запитів Solr звертається до індексу і виконує пошук та сортування за необхідними полями.

Окрім прив’язки до типів, поля мають ще й інші параметри, які називаються атрибутами. Якщо типи полів визначають дані, які зберігаються, то атрибути описують поведінку полів у системі.

Apache Solr підтримує велику кількість атрибутів, серед яких є основні наступні:

- Indexed – використання поля при пошуку.
- Stored – повернення поля у результаті пошуку.
- Required – чи обов’язкове поле для заповнення.
- MultiValued – чи може поле містити набір значень.
- OmitNorms – нормалізація довжини поля.
- DocValues – представлення поля у колонко-орієнтованій структурі (використовується при фасетингу, сортуванні та для функціональних запитів)

2.1.3 Схеми

Схема – це декларативний опис полів, їх типів, параметрів, а також різних обробників та фільтрів. У Apache Solr схеми описуються у форматі XML у спеціальному конфігураційному файлі.

Загальна структура схеми у Apache Solr виглядає наступним чином:

```
<schema>
  <types>
    <fieldType>
  <fields>
    <field>
  <copyField>
  <dynamicField>
  <similarity>
  <uniqueKey>
</schema>
```

Кореневою директивою є schema, в якій перелічуються усі інші директиви налаштування:

- Типи
- Поля
- Копійовані поля
- Динамічні поля
- Директива similarity, яка використовується для оцінювання документів про пошуку
- Унікальний ключ

У Apache Solr налаштування схеми зберігаються у окремому файлі для кожного окремого ядра. У ранніх версіях структура схеми зазначалась вручну у файлі schema.xml, але з розвитком системи стало можливим внесення змін у схему через так званий Schema API. При цьому зміни вносяться автоматично у окремий файл managed-schema.

У другому підході вважається небажаним ручна модифікація файлу, хоча це можливо. Перевагою використання Schema API є те, що схема може змінюватись у режимі реального часу під час роботи системи, тоді як внесення змін у файл schema.xml вимагає перезавантаження сервісу Solr.

2.1.4 Режим Schemaless

У нових версіях розробники додали можливість використовувати режим schemaless, який не вимагає від адміністраторів системи явно задавати схему та прописувати поля. Фактично для початку роботи потрібно просто внести необхідні документи, а система автоматично просканує усі наявні у документах поля, визначить їх типи та призначить їм атрибути (чи потрібна нормалізація, чи може міститись декілька значень у полі тощо). Усі невідомі поля будуть також автоматично вноситись у схему, але при цьому адміністратор може в ручному порядку видалити або виправити їх.

Очевидним недоліком є повний автоматизм, який не завжди спрацьовує правильно і точно. Розробники Apache Solr зазначають, що цей режим не слід використовувати на продакшені.

2.1.5 Копійовані та динамічні поля

Окрім звичайних полів у Apache Solr присутні ще два типи, які значно розширюють можливості для конструювання схем: копійовані поля (CopyFields) та динамічні поля (DynamicFields).

Копійовані поля – це механізм для поліморфного збереження даних. Фактично цей тип являє собою правило, яке ставить у відповідність одне поле до іншого, копіюючи його вміст перед виконанням перетворень. Таким чином за допомогою копійованих полів можна зберігати одні дані у різних формах, а також робити складені поля (поля, які містять в собі декілька інших).

Динамічні поля дозволяють генерувати звичайні поля неявно за допомогою заданих макетів. У атрибутах динамічного поля вказується фільтр для імен полів, до яких потрібно застосувати певний тип та параметри аналізу. Наприклад, динамічне поле

```
<dynamicField name="*_str" type="string" indexed="true" stored="true"/>
```

створить для вхідних полів first_name_str, last_name_str відповідні поля з типом string, та атрибутами indexed і stored встановленими у значення true.

Ці два корисних інструменти Solr дозволяють гнучко конструювати схеми. Приклади їх використання наведені у розділі 3.

2.1.6 Токенізатори та фільтри

Токенізатори і фільтри є інструментами аналізу у Apache Solr.

Токенізатор – обробник, який відповідає за розбиття вхідних даних на токени. Залежно від цілей, використовуються різні токенізатори. Так, наприклад, для розбиття послідовності по пробілам існує WhitespaceTokenizator. Для ігнорування розділових знаків існують ClassicTokenizer та StandardTokenizer. Для збереження цілої фрази використовують KeywordTokenizer. Розбиття на N-грамми реалізовують NGramTokenizer та EdgeNGramTokenizer.

Існують і більш специфічні токенізатори, які дозволяють обробляти файлові шляхи та URL, застосовувати регулярні вирази та навіть виокремлювати семантичні конструкції.

В аналізі поля спочатку використовується токенізатор, а потім до результату застосовуються фільтри, які можуть змінювати утворені токени, додавати нові або фільтрувати за заданим правилом.

Наприклад, за допомогою фільтрів можна реалізувати заміну символів на ASCII, щоб позбутися складнощів при пошуку; привести усі токени в нижній регістр; застосувати стемінг чи видалити дуплікати.

Слід зазначити, що ці два інструменти можуть бути застосовані як при індексуванні даних, так і при обробці запитів.

2.2. Python

Для маніпуляцій над даними та обгортанням API пошукової платформи необхідно розробити відповідне ПЗ. Для розробки було обрано мову програмування Python, яка є інтерпретованою мовою загального призначення. Серед основних причин вибору Python це швидкість розробки та великий набір супутніх бібліотек, які дозволяють легко та компактно вирішувати велику кількість завдань.

2.2.1 pysolr

Для взаємодії з сервером Apache Solr було обрано pysolr – клієнт, що написаний на Python. Він містить основні можливості для додавання, зміни та видалення різних видів полів.

Pysolr є чи не єдиним клієнтом для Python, який підтримується. При цьому його застосування обмежене і не підтримує ряд можливостей, які доступні через доступ до Schema API напряму. Саме тому паралельно з цією бібліотекою було вирішено використовувати бібліотеку requests для досягнення відсутніх у pysolr функцій.

2.2.2 FastAPI

Для реалізації API мова Python має величезну кількість різноманітних веб-фреймворків. Було розглянуто декілька варіантів, а саме FastAPI, Flask та Bottle, які належать до класу “легких”.

Проаналізувавши переваги та недоліки кожного з них було обрано веб-фреймворк FastAPI. За наявними бенчмарками він значно переважає Flask та Bottle у швидкості обробки запитів [11]. Також визначальним фактором була швидкість розробки та простота використання.

Серед інших переваг FastAPI можна виділити:

- Зручну систему налаштувань.
- Автоматичну валідацію через типізацію, що значно скоротить кількість помилок при розробці, а також зменшить кількість написаного коду.
- Автоматичну генерацію документації на основі визначення функцій-хендлерів, типізації та докстрінгів мови Python.

Розділ 3. Реалізація системи

3.1 Розробка схеми

На самому початку розробки необхідно приблизно окреслити, як буде виглядати схема колекції, а саме визначити назви полів, їх типи та основні параметри, якими оперує Solr: індексація, зберігання, обов'язковість, багатозначність тощо. У таблиці 3.1 вказані основні поля, які будуть використані в проєкті.

Таблиця 3.1 – Поля схеми

Назва поля	**Тип**	***Indexed***	***Stored***	***Multivalued***	***OmitNorms***
id	string	+	+	–	+
artist_name	text	+	+	–	+
track_name	text	+	+	–	+
release_date	long	–	+	–	+
genre	string	+	+	+	+
lyrics	text	+	+	–	–

Усі поля тут, окрім id, будуть братися із вхідних даних (документів). Поле id, хоча і не є обов'язковим для заповнення, для кожного запису буде генеруватися автоматично за допомогою налаштування у solrconfig.xml:

```
<!-- ID auto generation -->
<updateRequestProcessorChain>
  <processor class="solr.UUIDUpdateProcessorFactory">
    <str name="fieldName">id</str>
  </processor>
```

```
    <processor class="solr.LogUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
  </updateRequestProcessorChain>
```

Як було описано у розділі 2, різниця між типами string та text полягає у наявності токенізації та можливості часткового пошуку в останньому. Так, по полю id немає необхідності виконувати пошук, а поле genre має обмежену кількість можливих значень, які у стримінгових сервісах, скоріш за все є enum-ами, тому пошук по точному співпадінню є цілком задовільним.

Також поле genre єдине поміж усіх інших має встановлений параметр multiValued, що необхідно для можливості вказування кількох типів жанрів для однієї композиції.

Поле release_date має тип long, так як у даних, що використовуються для розробки, вказаний лише рік оприлюднення композиції.

Параметр omitNorms, як вже було зазначено у розділі 2, відповідає за вимкнення нормалізації довжини полів. У документації Apache Solr вказано, що вимкнення нормалізації має місце лише для повнотекстових полів. Саме тому його було встановлено у false для lyrics.

3.2 Вхідні дані для тестування

Для тестування пошукового функціоналу системи було використано набір даних “Music Topics and Metadata” [12], який містить понад 25 тисяч записів, які містять детальну інформацію про музичні композиції з 1950 до 2019 року.

Окрім основних даних про назву пісні та виконавця, датасет містить частини текстів цих пісень, жанр, рік релізу. Також наявні поля, які описують параметри звуку, але в рамках даної курсової роботи вони використовуватись не будуть.

3.3 Опис функціональних особливостей

3.3.1 Фільтрація зайвих даних

При індексації документів дуже часто виникає ситуація, коли у систему потрапляють зайві дані. Під час розробки ця проблема виникла на практиці.

Schemaless режим за замовчуванням містить налаштування для автоматичного додавання усіх невідомих полів, які надходять разом з даними. Як і зазначалось у пункті 2.1, цей підхід має суттєвий недолік у тому, що таким чином в схему потраплятимуть поля, які не важливі для роботи системи.

Також при використанні різних джерел (документів), в яких наявна деяка частина полів, що відрізняються (наприклад, метадані), в системі різні записи матимуть різні заповнені поля, що є небажаним.

Саме для цього було вирішено додати жорстку фільтрацію полів при індексації. У Apache Solr це можливо зробити за допомогою спеціального типу ignored та створення динамічного поля “*”.

```
<fieldType
     name="ignored"
```

```
        class="solr.StrField"
        indexed="false"
        stored="false"
        multiValued="true"/>
    <dynamicField
        name="*"
        type="ignored"
        multiValued="true"
        indexed="false"
        stored="false"/>
```

Атрибути indexed та stored для типу ignored встановлені у значення False, тому ні для пошуку, ні для виведення результату поля із вказаним типом ignored не будуть використовуватись. В свою чергу динамічне поле буде перехоплювати та встановлювати цей тип на будь-які невідомі для системи поля.

3.3.2 Вилучення дублікатів

Внесення пісень у пошукову базу може супроводжуватись виникненням дублікатів записів. Solr дозволяє зручно уникнути цієї проблеми за допомогою генерації хешів на основі деяких полів.

```
    <updateRequestProcessorChain name="dedupe">
      <processor
class="solr.processor.SignatureUpdateProcessorFactory">
        <bool name="enabled">true</bool>
        <str name="signatureField">id</str>
        <bool name="overwriteDupes">true</bool>
        <str name="fields">
```

```
            artist_name,track_name,release_date
        </str>
        <str name="signatureClass">
            solr.processor.Lookup3Signature
        </str>
    </processor>
    <processor class="solr.LogUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
</updateRequestProcessorChain>
```

3.3.3 Уніфікація пошуку

Заради спрощення пошукової системи для кінцевих користувачів зручніше надати одну єдину кінцеву точку взаємодії. У більшості сучасних музичних сервісах, як і інших системах типу YouTube чи Google, є одне поле, в яке користувач вводить свій запит. При цьому пошук виконується по усім можливим атрибутам, а не по якомусь конкретному.

Для реалізації такого функціоналу у Apache Solr існують так звані CopyField. Вони записуються у формі правил встановлення відповідності між полями.

Кінцевою точкою взаємодії у системі обрано текстове поле search. Воно повинне бути прибране з результатів, але доступне для індексації, щоб система могла виконувати пошук по ньому. Для направлення пошукових запитів з інших полів на поле search, додамо відповідні CopyField:

```
    <field name="search" type="text_general" indexed="true"
stored="false"/>
    <copyField source="artist_name" dest="search"/>
    <copyField source="lyrics" dest="search"/>
    <copyField source="track_name" dest="search"/>
```

Більшості користувачів для навігації по музичним композиціям достатньо можливості пошуку по назвам пісень, тексту та виконавцям.

Також для зручності поле search було встановлено за замовчуванням при відправці пошукових запитів на сервер. Тож тепер, якщо не вказувати конкретне поле пошуку, Apache Solr автоматично буде підставляти search.

3.3.4 Нечіткий пошук

Як відомо, користувачі під час взаємодії з будь-якими системами не завжди слідують рекомендаціям, а також можуть робити те, що не передбачено системою, чи не зовсім так як це бачили розробники.

Під час формулювання запиту користувач може допускати помилки, переставляти слова місцями або пропускати їх. Також, так як у нашій системі передбачений пошук по текстам пісень, потрібно враховувати той факт, що у реальний ситуаціях такі запити будуть вкрай неточними.

Для вирішення цієї проблеми використовують такий підхід як нечіткий пошук (fuzzy search). Apache Solr має вбудовану підтримку для цього.

Одним з найпростіших варіантів є використання спеціального символу “~” у кінці слів запиту. Також після цього символу можна вказати

максимальну відстань від введеного слова до тих, що можуть потрапити у запит. Щоб зробити нечіткий пошук по кільком словам, потрібно вказати оператор нечіткого пошуку після кожного слова в запиті, що може виявитися не дуже зручним для ручного використання, але дуже просто вирішується програмно.

Інший підхід передбачає використання N-грамного фільтра, але від цього способу було вирішено відмовитись через деякі складнощі з налаштуванням у Apache Solr.

Враховуючи це, було вирішено реалізувати підстановку fuzzy операторів засобами мови Python.

3.3.5 Кросмовність

Слід враховувати, що пісні можуть бути написані не лише англійською мовою. Багато європейських мов мають свої специфічні символи, які складно піддаються пошуку, враховуючи той факт, що користувачі про формуванні запитів можуть змінювати їх на аналогічні латинські.

Одним із варіантів вирішення цієї проблеми є збереження полів з використанням ASCII кодування, замінюючи нестандартні символи різних мов на латинські аналоги (фільтр ASCIIFoldingFilterFactory). При цьому не можна видаляти оригінальний запис. Саме для цього у Apache Solr існує CopyField. При внесенні назви виконавця, пісні або її тексту, було вирішено створювати додаткове поле з приставкою _proc, яке буде використовуватись для саме для пошуку і не відображатиметься в

результатах. Для відображення користувачу будуть використовуватись оригінальні поля.

Обробка запиту буде містити приведення токенів до ASCII кодування, після чого буде здійснюватись пошук полями з приставкою _proc.

3.3.6 Маппінг полів

Так як система створена для інтеграції з іншими сервісами, необхідно вирішити питання, як саме дані будуть потрапляти у систему. Одна із очевидних потреб це встановлення відповідності полів даних зовнішніх сервісів і пошукової платформи.

Відтак для цього засобами мови Python було реалізовано функцію, яка приймає CSV файл як аргумент, а також словник відповідності полів з зовнішнього сервісу до полів, які використовуються системою. В результаті обробки повертається новий файл зі зміненими полями, який вже далі індексує Apache Solr.

3.4 Архітектура системи

Після завершення розробки власне пошукової частини, необхідно реалізувати допоміжні функції, які описані у пункті 3.3, а також засоби комунікації з зовнішніми сервісами. Як було зазначено під час вибору

інструментів у розділі 2, для цих цілей використано мову програмування Python.

Розглянемо детальніше архітектуру системи, зображену на рис. 3.1.

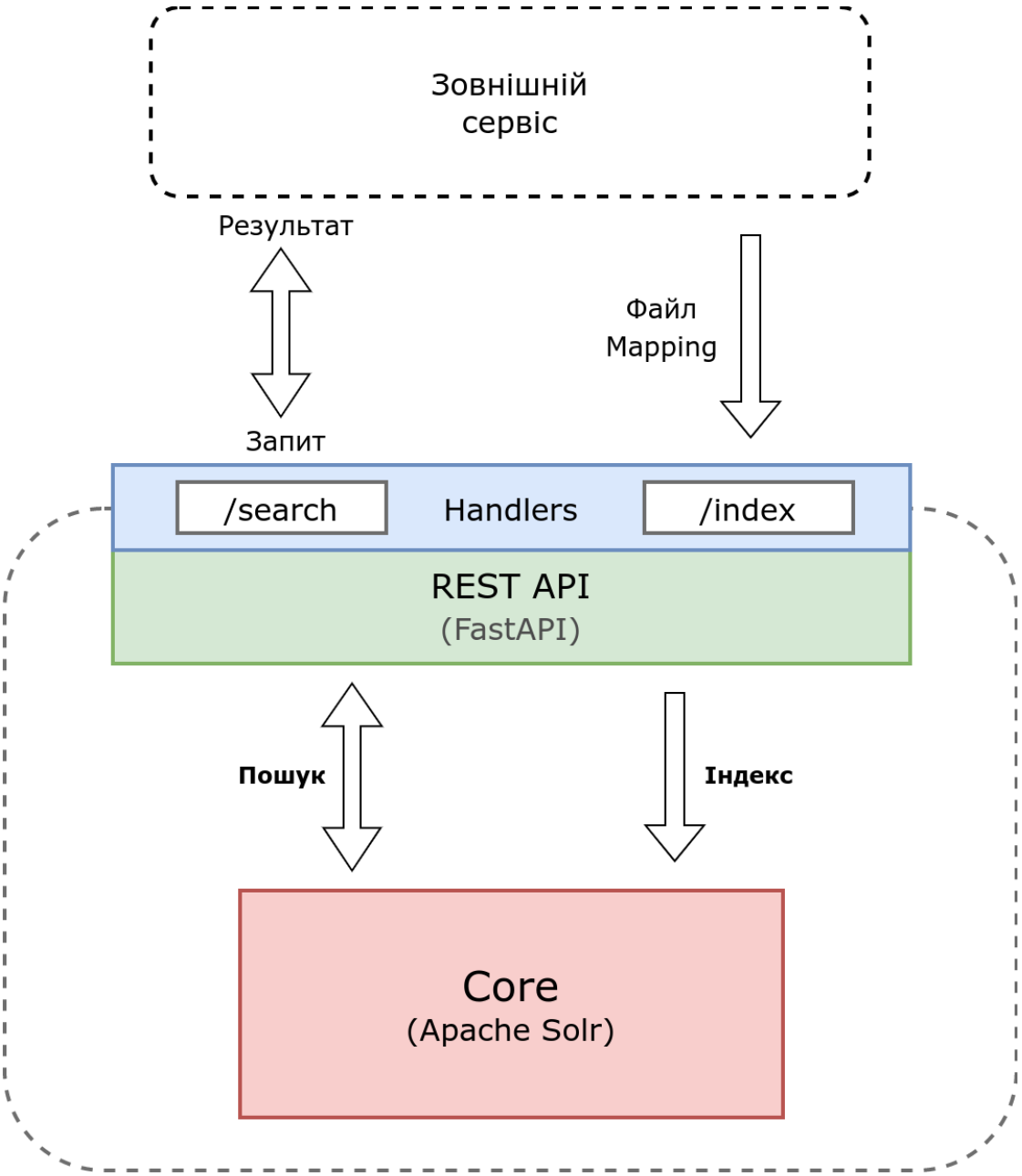

Рисунок 3.1 – Архітектура пошукової системи

Основними функціональними частинами системи є:

- Ядро, яке виконує пошукові функції – Apache Solr.
- Веб-сервер, який надає REST API для взаємодії з ядром.

Зовнішній сервіс має два ключові ендпоінти, за допомогою яких виконується пошук та додавання нових документів у систему:

- GET /search
- POST /index

Веб-сервер отримує запити ззовні, формує на їх основі запити до Apache Solr, виконуючи додаткову обробку даних (наприклад, маппінг з пп. 3.3.6), і після отримання результату повертає його, також попередньо здійснюючи обробку.

3.5 Формати запитів та відповідей

Список можливих параметрів до едпоінтів показаний у таблиці 3.2. Обов’язкові параметри виділені напівжирним шрифтом. Також слід зазначити, що для /search параметри передаються через URL-стрічку, а для /update – через POST-параметри.

Таблиця 3.2 – Параметри ендпоінтів системи

Endpoint	Параметри	Опис	За замовчуванням
/search GET	**query: string**	Пошуковий запит	—
	start: int	Параметр, який позначає, скільки знайдених рядків треба пропустити перед поверненням результату	0

Продовження таблиці 3.2

	row_num: int	Параметр, який позначає, скільки рядків потрібно повернути в результаті	10
	fuzzy: bool	Чи потрібно використовувати fuzzy search	True
/update POST	**update_file: file**	Файл з записами, які потрібно проіндексувати	—
	mapping_file: file	JSON файл, який містить маппінг полів	None

У таблиці 3.3 показана структура відповіді для ендпоінтів.

Таблиця 3.3 – Формат відповіді для ендпоінтів системи

Endpoint	Параметри	Опис
/search GET	doc_found: int	Загальна кількість знайдених документів
	start: int	Параметр, який позначає, скільки знайдених рядків було пропущено в результаті
	end: int	Параметр, який позначає номер документа, що був повернутий останнім
	results: array	Масив зі знайденими документами
/update POST	result: str	Результат виконання операції. OK – якщо, успішно, інакше – повідомлення про помилку

3.6 Тестування програми

Для перевірки роботи програми було використано інструмент тестування API Postman. Було сформовано запити різних форматів для моделювання поведінки реального користувача, який звертається до системи для пошуку певної композиції. Розглянемо декілька прикладів:

1) Користувач знає точну назву виконавця та пісні.

Запит: “Three days grace chalk outline” (Three Days Grace – Chalk Outline)

Результат:


```
{
    "doc_found": 10352,
    "start": 0,
    "end": 10,
    "results": [
        {
            "artist_name": "three days grace",
            "track_name": "chalk outline",
            "release_date": 2012,
            "genre": [
                "pop"
            ],
             "lyrics": "curse cross beat ones open shatter ones think love leave like chalk outline sidewalk wait rain away away come scene crime dead speak leave leave lyric commercial",
            "id": "7c3dc2cc-72b4-4677-9b5a-768c531ede44",
            "_version_": 1735167976184741890
        },
        {
            "artist_name": "three days grace",
            "track_name": "over and over",
            "release_date": 2006,
            "genre": [
                "pop"
            ],
              "lyrics": "feel bring blame try away chase fall feel like stay drag pull away chase fall fall thoughts head
```

```
live time feel dead know best want instead waste time fall
fall",
            "id": "98d11df2-4b06-4dc9-9106-fe490241ee0c",
            "_version_": 1735167975984463873
        },
        {
            "artist_name": "three days grace",
            "track_name": "it's all over",
            "release_date": 2006,
            "genre": [
                "pop"
            ],
             "lyrics": "bottle know mind race needle break skin
scar sink begin edge fall edge fall know run blood vain mind
race get skin give begin edge fall edge fall dead inside
wonder dead inside wonder dead inside wonder edge fall dead
inside wonder edge fall dead inside wonder",
            "id": "5d8119e6-3562-47cc-997d-4ccbf8c6d724",
            "_version_": 1735167975965589505
        },
     ...
}
```


2) Користувач знає назву виконавця і декілька слів з пісні.

Запит: “lorde gold teeth grey trippin” (Lorde – Royals)

Результат:


```
{
    "doc_found": 14600,
    "start": 0,
    "end": 10,
    "results": [
        {
            "artist_name": "lorde",
            "track_name": "royals",
            "release_date": 2013,
            "genre": [
                "pop"
            ],
              "lyrics": "see flesh teeth wed ring movies proud
address tear postcode envy song like gold teeth grey trippin
bathroom bloodstains ball gown trashin hotel room care drive
cadillacs dream everybody like maybach diamonds timepiece plan
islands tiger gold leash care aren catch affair royals royals
blood kind luxe crave different kind buzz ruler ruler queen
```

```
baby rule rule rule rule live fantasy friends crack code count
dollars train party know know fine come money song like gold",
            "id": "f78be060-9115-4030-a152-ded2c316ec88",
            "_version_": 1735167976243462144
        },
        ...
}
```


3) Користувач знає приблизні слова з пісні.

Запит: “People like you when they on the mind” (Mac Miller – Objects in the Mirror)

Результат:


```
{
    "doc_found": 23909,
    "start": 0,
    "end": 10,
    "results": [
        {
            "artist_name": "mac miller",
            "track_name": "objects in the mirror",
            "release_date": 2013,
            "genre": [
                "pop"
            ],
             "lyrics": "people love you when they on your mind
a thought is love's currency And I been thinking about her all
the time I've never seen somebody put together perfectly What
would I have to do to call you mine Someone like you is so
hard to find you can open up your eyes or you can walk it
blind",
            "id": "3e0d51a6-fcc8-45b7-95f2-77e4b861bada",
            "_version_": 1735172202032005120
        },
        ...
}
```


4) Користувач вводить запит з помилками.

Запит: “tom oddel anozer lov“ (Tom Odell – Another Love)

Результат:

```
{
    "doc_found": 16174,
    "start": 0,
    "end": 10,
    "results": [
        {
            "artist_name": "the chordettes",
            "track_name": "come home to my arms",
            "release_date": 1957,
            "genre": [
                "pop"
            ],
               "lyrics": "sweetheart soldier handsome suddenly
send away patiently wait leave write say darling tell come
station jump train march double lover roses entwine arm arm
arm surrender soldier soldier parade order order obey duty
order hurry home come station jump train march double lover
roses entwine arm arm arm surrender girl love soldier cause
away soldier duty answer call loudest come station jump train
march double lover roses entwine arm arm arm surrender",
            "id": "bc678dd6-a921-4cf8-bb90-caa1efd7e0d1",
            "_version_": 1735172200353234944
        },
        {
            "artist_name": "tom odell",
            "track_name": "another love",
            "release_date": 2013,
            "genre": [
                "pop"
            ],
                  "lyrics": "wanna know care cold know bring
daffodils pretty string like spring wanna kiss feel right tire
share nights wanna wanna tear tear tear tear somebody hurt
wanna fight hand break time voice fuck rude word know lose
sing song sing heart wanna wanna learn tear tear tear tear
need heart think wanna sing song sing heart wanna wanna fall
tear tear tear tear",
            "id": "6825330d-cbc8-46ae-a16a-2406281ecda4",
            "_version_": 1735172202030956544
        },
     ...
}
```


Як видно, шукана пісня повертається у більшості випадків на першому місці. Результат останнього запиту розташував шукану пісню на другому місці. Загалом така поведінка задовільна, так як при виведенні

результатів користувач все одно побачить композицію, яку він шукає, у п’ятірці найбільш співпадаючих.

3.7 Можливості для покращення

Реальні системи працюють з мільйонами записів, тому результати пошуку можуть відрізнятися за точністю від тестового середовища. При практичному використанні цілком ймовірно, що необхідно буде підлаштувати Apache Solr. Наприклад, відкоригувати токенізатори та фільтри полів або змінити баланс ваг при пошуку.

Окрім цього на практиці, ймовірно, потрібно буде використати кластерний режим роботи – SolrCloud, для покращення відмовостійкості та розподілення навантаження.

Як і будь-яку іншу систему, розробку даної курсової роботи можна покращити та доповнити додатковими функціональними можливостями. Серед пунктів, які можна виділити:

- Додавання синхронізації із зовнішніми сховищами даних.
- Інтеграція з рекомендаційними системами (додавання динамічних ваг для улюблених жанрів та виконавців).
- Додавання фасетингу для підбивання статистики по окремим жанрам.
- Підтримка додаткових полів (наприклад, параметрів звуку).

Внутрішню реалізацію можна покращити за допомогою контейнеризації. Це дозволить імплементувати мікросервісну архітектуру в повній мірі, а також спростить експлуатацію та розгортання ПЗ.

Висновки

В результаті виконання курсової роботи було розроблено систему пошуку у каталогах музичних композицій для стримінгових сервісів. Як ядро системи було використано Apache Solr – open-source пошукову платформу, засновану на бібліотеці Apache Lucene, з широкими можливостями для повнотекстового пошуку.

Було досліджено можливості Apache Solr, проаналізовано архітектуру платформи. Також було розібрано основні концепції та визначення, притаманні для сфери повнотекстового пошуку. Окрім цього, було на практиці випробувано веб-фреймворк FastAPI мови Python, для написання програмного інтерфейсу для взаємодії зовнішніх сервісів з пошуковою системою.

До переваг Apache Solr можна віднести швидкість пошуку, велику кількість функціональних можливостей, підтримка кластеризації, можливість конфігурування через REST API.

У той же час під час розробки практичної частини виникали певні складнощі з налаштуванням, відлагодженням при виникненні помилок та з пошуком інформації у мережі інтернет стосовно питань, пов’язаних з Apache Solr. Ці негативні фактори слід розглядати під час вирішення питання, яку саме платформу застосувати для імплементації пошуку у власній комерційній системі, тому що це може суттєво вплинути на якість створеного рішення.

Загалом можна сказати, що Apache Solr займає своє місце у сфері пошукових платформ і використовується у різних проєктах, хоча й програє по багатьох пунктах альтернативній системі ElasticSearch.

Список використаної літератури

1. Spotify – About Spotify [Електронний ресурс] – Режим доступу до ресурсу: https://newsroom.spotify.com/company-info/
2. 5 things to try with Wear OS on the Samsung Galaxy Watch4 [Електронний ресурс] – Режим доступу до ресурсу: https://blog.google/products/wear-os/galaxy-watch4/
3. Introduction to Information Retrieval – Tokenization [Електронний ресурс] – Режим доступу до ресурсу: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html
4. Мелтон Д., Querying XML / Мелтон Д., Бакстон С. – Morgan Kaufmann, 2006. – 439 с.
5. PostgreSQL: Documentation: 14 – Chapter 12. Full Text Search [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/textsearch.html
6. Apache Lucene – Welcome to Apache Lucene[Електронний ресурс] – Режим доступу до ресурсу: https://lucene.apache.org/
7. Apache Solr - Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Apache_Solr
8. IntegratingSolr - Solr - Apache Solr Software [Електронний ресурс] – Режим доступу до ресурсу: https://cwiki.apache.org/confluence/display/solr/IntegratingSolr
9. Free and Open Search: The Creators of Elasticsearch, ELK & Kibana | Elastic [Електронний ресурс] – Режим доступу до ресурсу: https://www.elastic.co/

10. Solr Vs Elasticsearch: Which Search Engine is Better? [Електронний ресурс] – Режим доступу до ресурсу: https://serverguy.com/comparison/solr-vs-elasticsearch/
11. Round 20 results - TechEmpower Framework Benchmarks [Електронний ресурс] – Режим доступу до ресурсу: https://www.techempower.com/benchmarks/#section=data-r20&hw=ph&test=fortune&l=zijzen-sf
12. Music Topics and Metadata - Mendeley Data [Електронний ресурс] – Режим доступу до ресурсу: https://data.mendeley.com/datasets/3t9vbwxgr5/1

Додатки

Додаток А

(обов’язковий)

Частини конфігураційного файлу solrconfig.xml

```
  <requestHandler name="/select" class="solr.SearchHandler">
    <lst name="defaults">
      <str name="df">search</str>
      <str name="echoParams">explicit</str>
      <int name="rows">10</int>
    </lst>
  </requestHandler>

  <updateRequestProcessorChain>
    <processor class="solr.UUIDUpdateProcessorFactory">
      <str name="fieldName">id</str>
    </processor>

    <processor class="solr.LogUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
  </updateRequestProcessorChain>

  <updateRequestProcessorChain name="dedupe">
    <processor
class="solr.processor.SignatureUpdateProcessorFactory">
      <bool name="enabled">true</bool>
      <str name="signatureField">id</str>
      <bool name="overwriteDupes">true</bool>
      <str
name="fields">artist_name,track_name,release_date</str>
      <str
name="signatureClass">solr.processor.Lookup3Signature</str>
    </processor>
    <processor class="solr.LogUpdateProcessorFactory" />
    <processor class="solr.RunUpdateProcessorFactory" />
  </updateRequestProcessorChain>
```

Додаток Б

(обов’язковий)

Частини конфігураційного файлу managed-schema.xml

```
<fieldType name="fuzzy" class="solr.TextField"
positionIncrementGap="100">
    <analyzer>
      <tokenizer class="solr.StandardTokenizerFactory"/>
      <filter class="solr.LowerCaseFilterFactory"/>
      <filter class="solr.ASCIIFoldingFilterFactory"/>
      <filter class="solr.NGramFilterFactory" maxGramSize="5"
minGramSize="2"/>
    </analyzer>
  </fieldType>
<fieldType name="ignored" class="solr.StrField"
indexed="false" stored="false" multiValued="true"/>

<field name="artist_name" type="fuzzy" uninvertible="true"
omitNorms="true" indexed="true" stored="true"/>
<field name="genre" type="strings" uninvertible="true"
omitNorms="true" multiValued="true" indexed="true"
stored="true"/>
<field name="id" type="string" required="false"/>
<field name="lyrics" type="text_general" multiValued="false"
required="false"/>
<field name="release_date" type="plong" uninvertible="true"
omitNorms="true" indexed="true" stored="true"/>
 <field name="search" type="text_general" indexed="true"
stored="false"/>
<field name="track_name" type="fuzzy" uninvertible="true"
omitNorms="true" indexed="true" stored="true"/>

<dynamicField name="*" type="ignored" multiValued="true"
indexed="false" stored="false"/>

<copyField source="artist_name" dest="search"/>
<copyField source="lyrics" dest="search"/>
<copyField source="track_name" dest="search"/>
```

Додаток В

(обов'язковий)

Файл обробників запитів handlers.py

```
import csv
import json
from codecs import iterdecode
from typing import Union

from fastapi import FastAPI, UploadFile

from src.helpers import map_fields_bin
from src.solr import solr_index, solr_search, solr_truncate

app = FastAPI()


@app.get("/search")
async def search(query: str, start: int = 0, row_num: int =
10, fuzzy: bool = True):
    """Main handler for searching queries"""

    return solr_search(query, start=start, row_num=row_num,
fuzzy=fuzzy)


@app.post("/index")
async def index_document(
    upload_file: UploadFile, mapping_file: Union[UploadFile,
None] = None
):
    """Main handler for indexing documents"""

    # apply mapping if needed
    if mapping_file:
        mapping = json.load(mapping_file.file)
        map_fields_bin(upload_file.file, mapping)

    # parse file to list of dictionaries
    reader = csv.DictReader(iterdecode(upload_file.file,
encoding="utf-8"))
    data = list(reader)

    # send index request
    res_status = solr_index(data)
```

```
    return {"result": res_status}


@app.delete("/index")
async def delete_index():
    """Helper handler for deleting all documents from the core

    Should be used carefully!
    """
    res = solr_truncate()

    return {"result": res}
```

Додаток Г

(обов’язковий)

Файл допоміжних функцій helpers.py

```
from pathlib import Path
from typing import BinaryIO, Union


def fuzzy_query(query: str) -> str:
    """Genegrate fuzzy string

    Args:
        query (str): query string

    Returns:
        str: fuzzy string

    """
    return " ".join([f"{token}~" for token in query.split()])


def map_fields_path(target_file_path: Union[Path, str],
mapping: dict, separator=","):
    """Rename file columns according to the specified mapping

    Args:
        target_file_path (Path): File Path
        mapping (dict): Dict with column mapping
        separator (str, optional): Column separator. Defaults
to ",".

    """

    with open(target_file_path, "r+") as f:
        # read header and parse
        header_line = f.readline().strip()
        columns = header_line.split(separator)

        # read all the other lines
        other_lines = f.readlines()

        # mapping
        new_columns = []
        for col in columns:
            if col in mapping:
                col = mapping[col]
```

```
            new_columns.append(col)

        # insert new header
        new_column_line = f"{separator.join(new_columns)}\n"
        other_lines.insert(0, new_column_line)

        # set cursor to the beginning of the file
        f.seek(0)
        f.truncate()

        # rewrite file
        for line in other_lines:
            f.write(line)


def map_fields_bin(target_file: BinaryIO, mapping: dict,
separator=b","):
    """Rename file columns according to the specified mapping
for binary files

    Args:
        target_file (Path): target binary file
        mapping (dict): Dict with column mapping
        separator (str, optional): Column separator. Defaults
to ",".

    """

    # read header and parse
    header_line = target_file.readline().strip()
    columns = header_line.split(separator)

    # read all the other lines
    other_lines = target_file.readlines()

    # convert mapping to binary
    mapping = {k.encode(): v.encode() for k, v in
mapping.items()}

    # mapping
    new_columns = []
    for col in columns:
        if col in mapping:
            col = mapping[col]
        new_columns.append(col)

    # insert new header
    new_column_line = separator.join(new_columns) + b"\n"
    other_lines.insert(0, new_column_line)
```

```
# set cursor to the beginning of the file
target_file.seek(0)
target_file.truncate()

# rewrite file
for line in other_lines:
    target_file.write(line)
target_file.seek(0)
```

Додаток Д

(обов’язковий)

Файл функцій взаємодії з Apache Solr solr.py

```
from typing import List

import pysolr
import requests

from src.config import CORE, SOLR_URL
from src.helpers import fuzzy_query

solr = pysolr.Solr(f"{SOLR_URL}/{CORE}/", always_commit=True)


def solr_search(
    query: str,
    search_field: str = "search",
    start: int = 0,
    row_num: int = 10,
    fuzzy: bool = True,
) -> dict:
    """Indermediate function for querying solr server

    Args:
        query (str): query string
        search_field (str, optional): field to search.
Defaults to "search".
        start (int, optional): number of rows to skip.
Defaults to 0.
        row_num (int, optional): number of rows to return.
Defaults to 10.
        fuzzy (bool, optional): if searching should be fuzzy.
Defaults to True.

    Returns:
        dict: dict with results and additional stats

    """
    # construct fuzzy query
    if fuzzy:
        query = fuzzy_query(query)

    # send request to solr
    items = solr.search(query, **{"df": search_field, "start":
start, "rows": row_num})
```

```
    # return composed result
    return {
        "doc_found": items.hits,
        "start": start,
        "end": start + row_num,
        "results": items.docs,
    }


def solr_index(data: List[dict]) -> str:
    """Indermediate function for indexing documents

    Args:
        data (List[dict]): list of documents

    Returns:
        str: result status
    """
    try:
        solr.add(data)
    except pysolr.SolrError as err:
        res = str(err)
    else:
        res = "OK"

    return res


def solr_truncate() -> str:
    """Indermediate function for truncating core data

    WARNING!
        This function will delete all of your indexed data!
        Be careful when using it!

    Returns:
        str: _description_
    """

    try:
        requests.post(
            f"{SOLR_URL}/{CORE}/update?commit=true",
            headers={"Content-Type": "application/json"},
            data='{"delete": {"query": "*:*"}}',
        )
    except pysolr.SolrError as err:
        res = str(err)
    else:
```

```
    res = "OK"
return res
```