Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

РОЗРОБКА СЕРВІСУ ЗБЕРЕЖЕННЯ ТА НАДАННЯ ДОСТУПУ ДО ДОКУМЕНТІВ УНІВЕРСИТЕТУ

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” - 121

Керівник курсової роботи:

д.т.н., доцент, Глибовець А. М.

_____(підпис)

“_____” _____ 2021 р.

Виконала студентка

Старовойт А.В.

_____(підпис)

“_____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

доцент _____

„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенці Старовойт Аліні Володимирівні факультету інформатики 3-го курсу

ТЕМА: Розробка сервісу збереження та надання доступу до документів університету

Вихідні дані: Сервіс для збереження та пошуку документів університету

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

Розділ 1: Системи зберігання та пошуку

Розділ 2: Пошукові двигуни

Розділ 3: Предметна область

Розділ 4: Огляд стеку технологій

Висновки

Список використаної літератури

Дата видачі „___” _____ 2020 р.
Керівник _____ (підпис)
Завдання отримав _____ (підпис)

Тема: Розробка сервісу збереження та надання доступу до документів університету

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітки
1	Отримання теми курсової роботи	10.11.2020	
2	Огляд технічної літератури за темою роботи	30.11.2020	
3	Встановлення необхідного програмного забезпечення	02.12.2020	
4	Реалізація серверної частини застосунку	07.01.2021	
5	Написання основної частини курсової роботи	06.05.2021	
6	Перегляд змісту роботи з керівником	08.05.2021	
7	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	15.05.2021	
8	Створення слайдів для доповіді та написання доповіді	17.05.2021	
9	Захист роботи	17.05.2021	

Студентка Старовойт А.В.

Керівник Глибовець А. М.

“_____” _____

Зміст

АННОТАЦІЯ 4
РОЗДІЛ 1: АНАЛІЗ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА ПРЕДМЕНТОЇ ОБЛАСТІ 6
1.1 Аналіз завдання 6
1.2 Аналіз предметної області 6
РОЗДІЛ 2: ОСОБЛИВОСТІ ПОВНОТЕКСТОВОГО ПОШУКУ ТА ВІДОМІ ПОШУКОВІ ДВИГУНИ 10
2.1 Що таке повнотекстовий пошук? 10
2.2 Аналіз можливих рішень для зберігання та пошуку 11
2.2.1 PostgreSQL 11
2.2.2 Пошукові двигуни на основі Apache Lucene - Solr та Elasticsearch 12
РОЗДІЛ 3: ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ 15
3.1 Spring 15
3.2 Elasticsearch 17
РОЗДІЛ 4: ОПИС СИСТЕМИ 18
4.1 Розгортка програми 18
4.2 Структура проекту 19
4.2.1 Spring Data Elasticsearch 20
4.2.2 DocumentService 23
4.2.3 Пошук документа в Elasticsearch 24
Висновки 25
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 26

АННОТАЦІЯ

У процесі написання курсової роботи було створено сервер для зберігання та пошуку документів університету, а саме: дипломних та курсових робіт, навчальних планів, розкладів, наказів тощо. Для реалізації зберігання та пошуку документів було обрано технологію Elasticsearch. Сервер написаний на Java з використанням Spring Boot .

ВСТУП

Кажуть, «Час плинний, а Могилянка вічна». І дійсно, мій університет вже понад 400 років готує спеціалістів в різних сферах життя. Проте не можна заперечувати, що нові епохи ставлять нові виклики, університет має постійно змінюватися, аби не втрачати актуальності, адаптуватися до нових реалій життя.

З розвитком сучасних технологій та інтернету все важливішим стає представлення університету в мережі. Києво-Могилянська Академія не пасе задніх – існують веб-платформи з інформацією про університет, для дистанційного навчання, для запису на дисципліни. Проте є ще дуже багато сервісів, які університет може й повинен надавати онлайн. Пандемія, що почалася 2020 року й змусила всіх перейти у віртуальний світ, робить це як ніколи актуальним.

Ця курсова робота демонструє створення сервісу з збереження й надання доступу до документів університету. Це можуть бути наукові роботи, різноманітні накази, постанови, навчальні плани та інші документи. Створений сервіс не є самостійним продуктом, а може лише слугувати частиною архітектури веб-платформи НаУКМА. В роботі надано функціонал для збереження та пошуку таких документів.

Для написаня використано мову Java та фреймворк Spring Boot. Документи зберігаються в Elasticsearch, NoSQL базі даних та пошуковій машині.

РОЗДІЛ 1: АНАЛІЗ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА ПРЕДМЕНТОЇ ОБЛАСТІ

1.1 Аналіз завдання

Головна задача цієї курсової роботи – побудова сервісу для зберігання та надання доступу до документів університету. Користувачі повинні мати змогу шукати курсові та дипломні роботи, навчальні плани та накази. Пошук має бути швидким, релевантним, ми також мусимо мати змогу фільтрувати результати за певними критеріями. Важливо мати змогу використовувати повнотекстовий пошук(full-text search), щоб знайти документи, навіть якщо ми не знаємо їх точної назви. Наприклад, коли ми хочемо знайти курсові роботи по розробці ігор, то в результаті пошуку отримуємо ті курсові роботи, де слова «розробка» та «ігри» зустрічаються, при чому результати будуть відсортовані за частотою, з якою слова присутні в документі.

1.2 Аналіз предметної області

Завдання полягає в збереженні та наданні доступу до документів університету. Після кількох консультацій з співробітниками університету щодо їх потреб та побажань було укладено список документів, полів документів, а також права доступу до цих документів. Результат дослідження наведено нижче:

1. Курсова робота
    a. ID
    b. Назва
    c. Анотація
    d. ПІБ автора
    e. Факультет
    f. Рік навчання

g. Спеціальність

2. Дипломна робота
    a. ID
    b. Назва
    c. Анотація
    d. ПІБ автора
    e. Факультет
    f. Спеціальність
    g. ПІБ Керівника

3. Відгук на дипломну роботу
    a. ID дипломної роботи
    b. ПІБ
    c. Посада
    d. Вчене звання

4. Наказ по університету
    a. ID
    b. Назва
    c. Номер наказу
    d. Короткий опис

5. Наказ по факультету (в межах факультету)
    a. ID
    b. Назва
    c. Номер наказу
    d. Короткий опис
    e. Факультет

6. Положення
    a. ID
    b. Назва
    c. Номер версії
    d. Дата затвердження

7. Інші нормативні документи (статут, правила, тощо)
    a. ID
    b. Назва

c. Короткий опис

8. Силабус

a. ID
b. Галузь знань
c. спеціальність
d. Назва освітньої програми
e. Назва навчальної дисципліни
f. Факультет
g. Дата затвердження
h. Номер версії (освітня програма може змінюватися)

9. Освітня програма

a. ID
b. Галузь знань
c. спеціальність
d. Дата затвердження
e. Номер версії (освітня програма може змінюватися)

Оскільки в даній роботі не передбачена реалізація авторизації, документи будуть доступні для всіх користувачів сервера. Передбачається, що в готовій системі буде присутній окремий сервіс, що відповідає за авторизацію, аутентифікацію та права доступу користувачів.

Для кожного документу буде надана можливість створення, видалення та читання документа за ID. Також за допомогою пошукового двигуна для кожного документа є можливість повнотекстового пошуку по вибраним полям.

Відгук на дипломну роботу може бути тільки один, тож було вирішено зберігати його як вкладений об’єкт в базі даних, та як внутрішній клас дипломної роботи в Java. В зв’язку з цим додано додатковий функціонал – пошук відгуку за номером роботи, додавання, видалення та редагування відгуку в дипломній роботі.

Освітня програма та силабус можуть змінюватися, саме тому необхідно зберігати номер версії цих документів. Додано можливість

редагування цих документів. Ми зберігаємо тільки останню версію документу. При створенні номер версії встановлюється 1, при кожному редагуванні документу номер її автоматично збільшується.

РОЗДІЛ 2: ОСОБЛИВОСТІ ПОВНОТЕКСТОВОГО ПОШУКУ ТА ВІДОМІ ПОШУКОВІ ДВИГУНИ

2.1 Що таке повнотекстовий пошук?

Повнотекстовий пошук (або Full text search) дає можливість шукати в наборі документів за складними запитами та, за бажанням, сортувати ці документи за відповідністю. Найчастіше такий пошук відбувається за допомогою індексації документів. В найпростішому варіанті, запит – це набір слів, а відповідний йому документ має найбільшу кількість входжень термінів з запиту[1].

Звичайні SQL бази даних дають можливість пошуку на відповідність запиту за допомогою ключових слів LIKE, ILIKE тощо. Проте їх можливостей недостатньо, для забезпечення повноцінного сучасного пошуку по застосунку. Одні з головних причин:

- Такі запити не розуміють людської мови. Наприклад слова «сісти» та «сидіти» будуть сприйматися по різному, не кажучи вже про синонімічні ряди.
- Вони не сортують результати за відповідністю. При наявності досить великого набору документів, це стає необхідністю
- Такий процес дуже повільний, адже для кожного запиту БД має перебрати всі документи.

На щастя, існує багато інструментів та застосунків, які підтримують повнотекстовий пошук. Ключовою фазою для надання можливості швидкого та ефективного пошуку є індексація документів. Імплементація кожної конкретної пошукової системи може різнитися, але основні кроки обробки документа(чи запиту) такі:

1. Розбиття документа(запита) на токени – окремі слова.

2. Приведення токенів до лексем. В цьому процесі спільнокореневі слова приводяться до єдиного зразка, прибираються закінчення, дефіси та великі літери.
3. Збереження обробленого та готового до пошуку документа. Зазвичай це масив, де зберігаються відсортовані лексеми та кількість входжень.

Як ми бачимо, процес індексації документів досить складний та тривалий, а також вимагає певних затрат пам’яті. Звичайно, виконання складних запитів пошуку тепер буде швидшим та еластичнішим, проте додавання та змінення документів займе набагато більше часу.

Повнотекстовий пошук використовується майже в кожному сучасному онлайн-застосунку, починаючи від гуглу й закінчуючи різноманітними магазинами. Алгоритми пошуку постійно вдосконалюються, а різноманітні пошукові системи готові запропонувати широкий вибір функціоналу на будь-який смак та вимоги.

В наступному розділі буде розглянуто 3 популярні пошукові системи, їх переваги й недоліки, а також доцільність використання кожної з них для поставленої задачі.

2.2 Аналіз можливих рішень для зберігання та пошуку

Популярність та необхідність повнотекстового пошуку призвела до великої кількості різноманітних систем та рішень. Кожна пошукова система має свої переваги та недоліки, тому головне для правильного вибору – дослідження предметної області.

2.2.1 PostgreSQL

PostgreSQL – це система контролю базами даних (СКБД) з відкритим кодом, написана на мові C. Postgres – одна з провідних безкоштовних реляційних баз даних. Вона надійна, швидка та легко розширюється. Оскільки це SQL база даних, інформація зберігається в таблицях, а таке рішення ідеальне для складних зв'язків, частої зміни та оновлення даних та транзакцій.

До того ж, Postgres надає можливості повнотекстового пошуку[1]. За допомогою SQL запиту ми можемо сформувати документ(єдину сутність), яку легко проіндексувати та використовувати для пошуку. Бібліотека дозволяє використовувати складні запити, автодоповнення тексту, сортувати документи за відповідністю до запиту та багато чого іншого.

Таким чином, якщо нам потрібні транзакції та реляційні зв'язки, Postgres дозволяє не додавати окрему систему пошуку, а зберегти дані в одному місці. Швидкість пошуку може бути більшою ніж в інших пошукових системах, проте це забезпечить цілісність даних та простоту архітектури.

2.2.2 Пошукові двигуни на основі Apache Lucene - Solr та Elasticsearch

Solr та Elasticsearch - пошукові системи з відкритим кодом. Обидві системи побудовані на Apache Lucene – бібліотеці для повнотекстового пошуку, написаній на Java[2]. І хоч функціонал в цих системах подібний, вони суттєво відрізняються в застосуванні, масштабуванні та типами запитів.

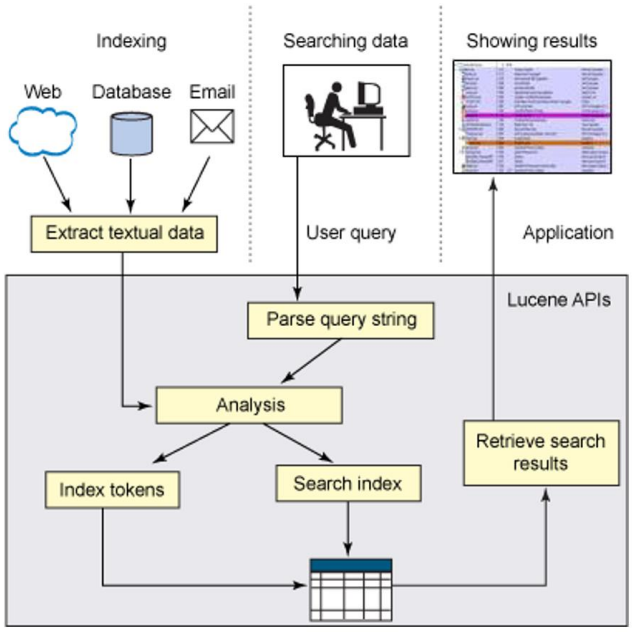

Рисунок 2.2.2.1: алгоритм роботи з бібліотекою Lucene[3]

На рисунку 2.2.2.1 схематично зображено алгоритм роботи з Apache Lucene. Текстову інформацію з бази даних, так само як і запит користувача, розбивають на токени, аналізують. Далі будується індекс - або для зберігання (додати документ до бази), або для пошуку(здійснити пошук в веб-застосунку. Якщо пошук є успішним, то застосунок отримує відповідь.

Розглянемо спільні якості пошукових двигунів Solr та Elasticsearch. Обидві системи легко встановити та інтегрувати. Вони мають багаточисельну спільноту послідовників(Solr – багаторічних відданих користувачів, Elasticsearch – молодих активних послідовників, кількість яких стрімко зростає). Для більшості широко використовуваних функцій підходять обидві системи. Побудовані на одній бібліотеці, вони надають майже однакові можливості в текстовому пошуку в режимі реального часу . Проте кожна система також має свої особливості, переваги та недоліки.

Solr розрахований на підприємства, та їх потреби інформаційного пошуку. Він також підтримує інтеграцію з багатьма інструментами для великих даних, таких як Hadoop та Spark. Solr є кращим вибором, якщо

потрібно часто працювати з Rich Text Format (RTF, «формат збагаченого тексту») документами, такими як Word, PDF та Excel.

Solr вийшов в 2004 році, натомість Elasticsearch було засновано 2010 року, і основним акцентом була здатність до масштабування. Як натякає нам назва, він є, власне, еластичним. Elasticsearch легко розподіляє данні на декількох доступних серверах. Він підтримує автоматичний менеджмент кластерів через вбудований модуль, тож програмісту не потрібно витрачати час на управління складними системами. В Solr зміни в кластерах – складний процес, що вирішується в ручному режимі[4]. Таким чином, в плані масштабування та автоматизації він програє – багато процесів потребують ручного налаштування та встановлення додаткових модулів.

Elasticsearch фокусується на масштабуванні, аналізі даних та знаходженні логічних кореляцій. Разом з Logstash (інструмент для обробки логів) та Kibana(інтерфейс для візуалізації даних), Elasticsearch формує ELK stack – потужне програмне забезпечення для зберігання, пошуку та аналізу даних.

Elasticsearch базується на JSON форматі. Це особливо вигідно для веб-застосунків, які обмінюються даними саме в цьому форматі.

Для нових застосунків, що з самого початку орієнтуються на Elasticsearch, дуже корисною є можливість використовувати його як повноцінну NoSQL базу даних – основне сховище даних застосунку. Для великих та складних систем проблемою може бути брак підтримки зв’язків та транзакцій, проте в такому випадку можна використати Elasticsearch як систему пошуку, побудовану на будь якій альтернативній базі даних.

РОЗДІЛ 3: ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

В цьому розділі буде оглянуто використані технології. Для написання сервера обрано фреймворк Spring, а для зберігання та пошуку – двигун Elasticsearch.

3.1 Spring

Spring – це сучасний популярний фреймворк для створення застосунків на Java. Його використання дозволяє значно спростити написання програм, автоматизувати стандартні процеси, аби зосередитися на справді важливих речах. Одним з головних підходів цього фреймворку, який спрощує життя програмістів є інверсія управління, або Inversion of Control(IoC).

Раніше програміст повинен був сам займатися створенням та зв’язуванням об’єктів та компонентів. У Spring натомість використовують контейнер Spring Container. Для керування компонентами він використовує ін’єкцію залежностей (Dependancy Injection). Всі об’єкти, які позначені як компоненти (Sprong Beans), контейнер автоматично створює та робить готовими для використання - завантажує необхідні класи в пам’ять, створює об’єкти та прив’язує їх до відповідних компонентів[5].

Для збірки компонентів Spring використовує конфігураційні XML файли, або, пізніше, Java код[6]. З плином часу програмісти відчули громіздкість цього підходу (велика кількість додаткових конфігураційних файлів, що обтяжують проєкт та ускладнюють розробку). Тому для полегшення роботи з компонентами було створено Spring Boot. Його основні властивості:

- Анотації замість ХМЛ
- Автоконфігурація (та сама «магія», яка відбувається непомітно для користувача)

- Convention over configuration принцип програмування ("угода головніша за конфігурацію"). Загалом полягає в тому, що замість наявної конфігурації використовується неявна «угода».

Розберемо детальніше анотації. Для того, щоб фреймворк зрозумів, що ми використовуємо компонент, нам потрібно всього лише поставити відповідну анотацію перед оголошенням класу - @Component, або дочірні для нього @Repository, @Controller, @Service. Інший спосіб – створити метод з анотацією @Bean, який повертає нам об’єкт потрібного нам класу. Для створення компонентів, якщо не вказано інакше, Spring використовує патерн Singletone - створюється лише один об’єкт(компонент), який використовується в різних частинах програми. Для використання компонентів, які були для нас створені, є анотація @Autowired та Dependency Injection.

Розберемо анотацію @SpringBootApplication, яка стоїть перед головним класом нашої програми та відповідає за значну частину «магії» SpringBoot. Насправді вона складається з трьох інших анотацій[8]:

@SpringBootApplication =

@EnableAutoConfiguration +

@ComponentScan +

@SpringBootConfiguration

@EnableAutoConfiguration – створює стандартні конфігураційні компоненти фреймворку, які зазвичай підходять для стандартного застосунку. Spring сам обирає необхідні компоненти, базуючись на залежностях, які вказані в програмі.[7]

@ComponentScan – сканує всі компоненти у цьому та всіх дочірніх пакетах, створює необхідні об’єкти

@SpringBootConfiguration – показує, що клас містить в собі конфігурацію застосунку

Таким чином бачимо, що «магії» не відбувається, натомість ми отримуємо зручний та безпечний фреймворк, який дозволяє уникнути непотрібного коду та зосередитися на необхідний речах.

3.2 Elasticsearch

В попередньому розділі було проаналізовано різні пошукові двигуни, їх сильні та слабкі сторони. Для роботи над цим сервером було обрано Elasticsearch з наступних причин:

1. Швидкість та масштабованість
2. NoSQL база даних для роботи з документами
3. Зручність формату JSON
4. Зручність інтеграції з Spring framework
5. Можливість використання як основного сховища даних

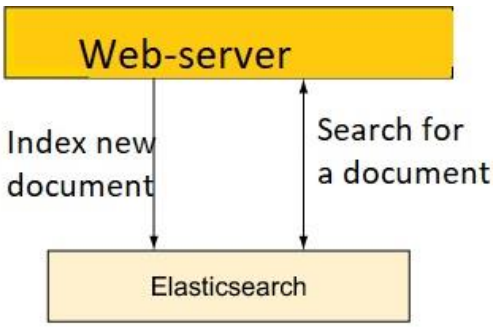

Рисунок 3.2.1 : модель комунікації бази даних та сервера

Оскільки ми обрали пошуковий двигун Elasticsearch, він буде і основним місцем сховища даних, нашою базою. Це рішення є одночасно простим та ефективним.

РОЗДІЛ 4: ОПИС СИСТЕМИ

4.1 Розгортка програми

Для розробки програми використано середовище розробки IntelijIdea. Для розгортки необхідно встановити на машині Elasticsearch та запустити його.

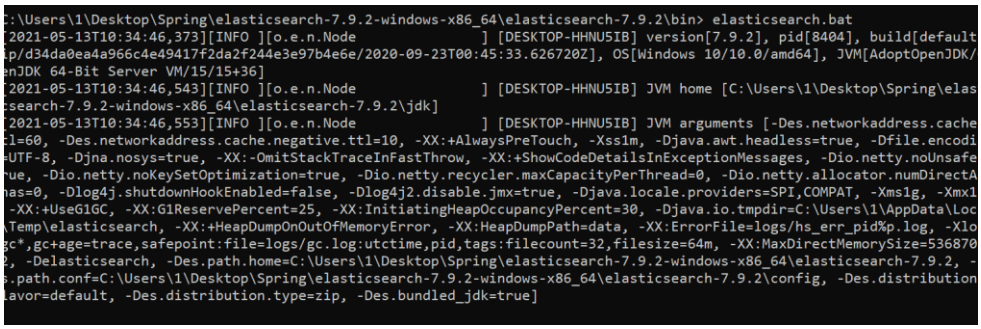

Рисунок 4.1.1 : запуск двигуна Elasticsearch з командного рядка

Важливим є те, щоб порт вказаний в конфігурації Elasticsearch та серверу застосунку збігалися. Я використовую стандартний порт 9200 для Elasticsearch та 8080 для нашого веб-сервера.

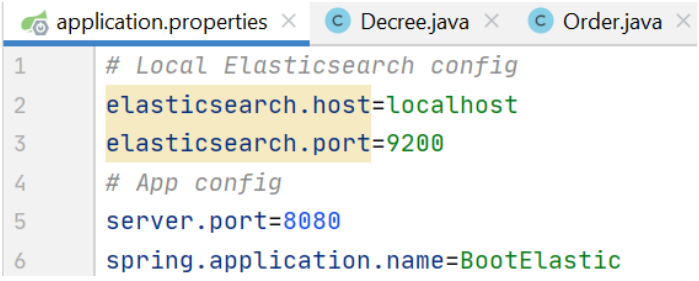

Рисунок 2.1.2 : конфігураційний файл веб-сервера з номерами портів

До списку залежностей програми необхідно додати Spring Data Elasticsearch

```
 <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
</dependency>
```

Таким чином при запуску застосунку, Spring фреймворк автоматично створить компонент, за допомогою якого буде відбуватися комунікація програми з базою даних та пошуковим двигуном.

4.2 Структура проекту

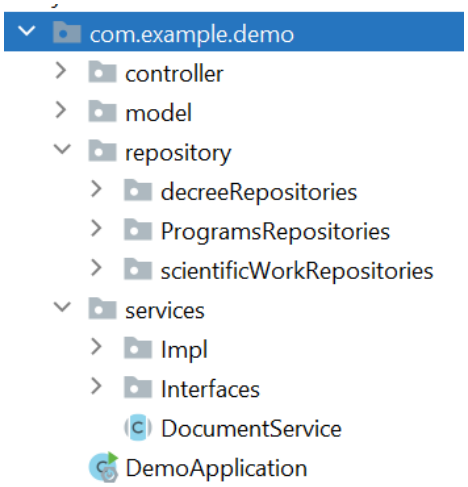

Рисунок 4.2.1: структура проекту

Структура проекту реалізує контролер-сервіс-сховище архітектуру. Контролер – це точка входу в застосунок. Для кожного документу визначено свій контролер. Кожен контролер залежить від компонента сервісу, де відбувається бізнес-логіка програми. Сервіс в свою чергу залежить від репозиторію, який під'єднаний до бази даних. В кожному з цих етапів для передачі даних використовується модель – Java-представлення об'єктів з якими ми працюємо. В цьому випадку, це документи – курсові роботи, накази, розклади тощо.

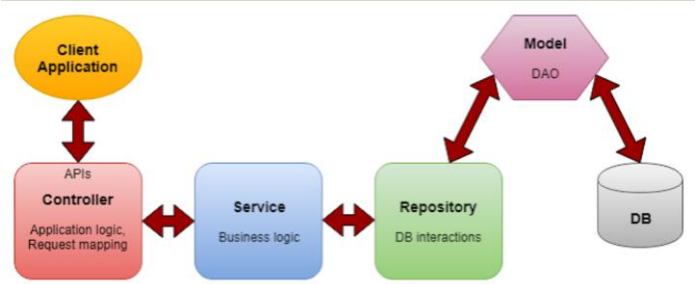

Рисунок 4.2.2: графічне представлення архітектури

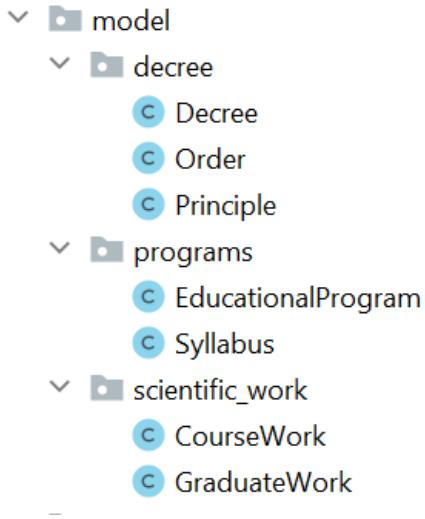

Рисунок 4.2.3: Моделі сутностей проекту

4.2.1 Spring Data Elasticsearch

Як було зазначено раніше, в сервері ми використовуємо Spring Data Elasticsearch. Цей проект дозволяє поєднати функціональність пошукового двигуна Elasticsearch та основних концепцій Spring[9].

Користувач може використовувати такі компоненти :

1. Темплейти(Templates) – абстракція для пошуку та індексації побудована на основі ElasticsearchRestClient
2. Сховища(Repository) – абстракція високого рівня, типова для фреймворку. Базові методи для роботи з даними Spring Repository

(створення, читання, пошук, видалення та редагування) створюються вже при оголошенні відповідного інтерфейсу.

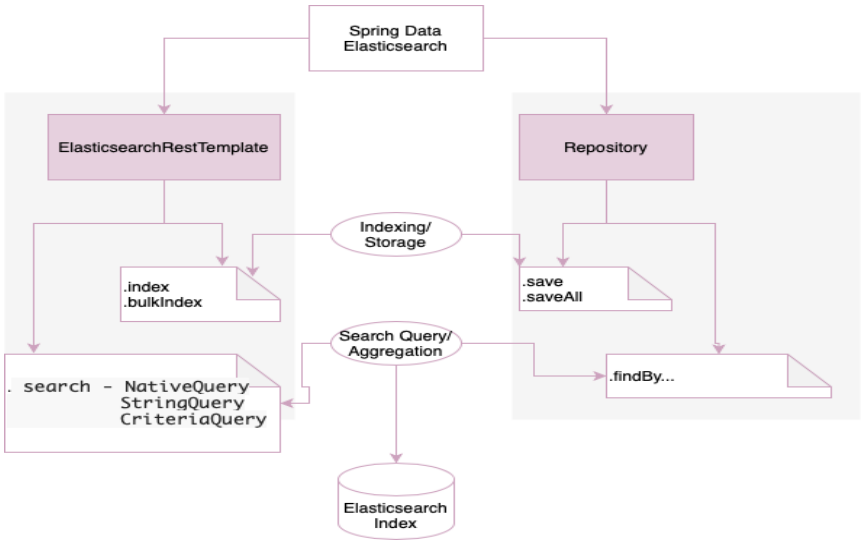

Рисунок 4.2.1.1: схема роботи Spring Data Elasticsearch

В даній роботі використано обидва підходи. Сховище - для створення, читання, збереження та видалення документів. Темплейт – для пошуку по різним полям.

Продемонструймо роботу сховища для одного з документів – курсової роботи

Створимо клас сутності(модель):

```java
@Document(indexName = "course-index")
@Data
public class CourseWork {
    @Id
    private String id;
    private String name;
    private String annotation;
    private String fullName;
    private int yearOfStudy;
    private String faculty;
    private String speciality;



    public static final Map<String, Float> searchFields=Map.of(
            k1: "name", v1: 1.0f,
            k2: "annotation",  v2: 1.0f
    );
}
```

Рисунок 4.2.1.2: клас сутності "Курсова робота"

Анотація *@Document* позначає, що даний клас є Elasticsearch документом, в сховищі буде створено відповідний індекс (indexName = „course-index”). За бажанням в анотацію можна додати інші параметри: кількість реплік, параметри щодо розподілення даних в сховищі, чи буде створюватися індекс автоматично при запуску застосунку тощо)[5]

Окрім стандартних полів опису документа, створено також статичне поле *„searchFields”* – за визначеними там полями відбувається пошук документа. Наприклад, для пошуку курсових робіт, до уваги береться відповідність запита назві та анотації курсової роботи.

Для створення сховища за допомогою Spring Data Elasticsearch, нам необхідно всього лише оголосити інтерфейс, що розширює ElasticsearchRepository, вказавши при цьому модель документа та тип ідентифікатора цієї моделі. Тепер ми можемо використовувати стандартні методи (створення, читання та видалення за ідентифікатором, редагування документу тощо), а також визначати власні(якщо методи досить прості та названі згідно контракту, то немає навіть потреби їх реалізовувати).

```
public interface CourseWorkRepository extends ElasticsearchRepository<CourseWork, String> {

    Collection<CourseWork> findByName(String text);

}
```

Рисунок 3: визначення сховища курсової роботи

4.2.2 DocumentService

Оскільки функціонал для всіх документів цього проекту подібний (створення, видалення, пошук по ідентифікатору, виведення всіх документів та повнотекстовий пошук) було прийнято рішення створити абстрактний клас «DocumentService» з реалізацією цих базових методів.

```
public abstract class DocumentService<T,R>  {

    private final ElasticsearchRepository<T,R> repository;
    private final ElasticsearchOperations elasticsearchRestTemplate;


    public DocumentService(ElasticsearchRepository<T, R> repository, ElasticsearchOp

    public T create(T document) {...}

    public Optional<T> read(R id) { return repository.findById(id); }

    public void delete(R id) { repository.deleteById(id); }

    public Iterable<T> findAll() { return repository.findAll(); }


    public Collection<T> search(String q, Map<String,Float> fields, Class<T> c, Stri
```

Рисунок 4.2.2.1: Абстрактний клас для документа

Кожен сервіс конкретного документу наслідує вищезазначений клас, отримуючи весь його функціонал з можливістю розширення.

4.2.3 Пошук документа в Elasticsearch

Продемонструємо пошук документу через ElasticsearchRestTemplate використовуючи NativeSearchQuery. Наша задача - скласти пошуковий запит з такими параметрами, щоб пошуковий двигун видавав документи в порядку їх відповідності до запиту. Elasticsearch з цією метою оцінює кожен документ.

```
public Collection<T> search(String q, Map<String,Float> fields, Class<T> c, String indexName){

    NativeSearchQuery searchQuery = new NativeSearchQueryBuilder()
            .withQuery(multiMatchQuery(q)
                    .fields(fields)
                    .type(MultiMatchQueryBuilder.Type.BEST_FIELDS))
            .build();


    return   search(searchQuery,c,indexName);
}

protected List<T> search(Query query, Class<T> c, String indexName) {
    final SearchHits<T> result = elasticsearchRestTemplate.search(query, c, IndexCoordinates.of(indexName));
    if (result.isEmpty()) {
        return Collections.emptyList();
    }
    final List<T> links = result.getSearchHits().stream().map(SearchHit::getContent).collect(Collectors.toList());
    return links;
}
```

Рисунок 4.2.3.1: методи, за допомогою яких відбувається пошук в документах

Для того, щоб шукати в вибраних полях (назва, анотація) використовується multiMatchQuery, та передані параметром fields поля. Важливо, що ми обираємо стратегію «найкращі поля». Вона передбачає, що загальна оцінка документу буде дорівнювати максимальній оцінці відповідності поля[6]

В другий метод ми передаємо вже сформований запит, клас та назву індексу. Саме він робить запит на сервер бази даних та повертає колекцію документів в порядку спадання відповідності.

Висновки

Метою цієї курсової роботи є розробка сервісу для зберігання та пошуку інформації про документи університету. В ході дослідження проаналізовано сучасні методи розробки, завдяки чому обрано фреймворк Spring Boot та пошуковий двигун Elasticsearch. Дані технології є сучасними, багатофункціональними та популярними на ринку.

Було розроблено сервіс з такими функціями, як: збереження, пошук, редагування та видалення для більш ніж 10 типів документів. Завдяки пошуковому двигуну Elasticsearch пошук документів є швидким та ефективним. Двигун дуже легко масштабується горизонтально, тому є можливість збереження великої кількості документів без втрати ефективності. Використана архітектура дає можливості для розширення функціоналу в подальшому.

В наступних роботах в першу чергу планується створення сервісу з авторизацією користувачів, для забезпечення безпеки. Також необхідне створення користувацького інтерфейсу, аби перетворити даний сервер в повноцінний продукт.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. PostgreSQL documentation: Chapter 12. Full text search [Електронний ресурс] https://www.postgresql.org/docs/current/textsearch-intro.html
2. Asaf Yial, 2020 - Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine? [Електронний ресурс] https://logz.io/blog/solr-vs-elasticsearch/
3. Isil Karabey Aksakalli , uğut kılıç, October 2016: "Comparison of Solr and Elasticsearch Among Popular Full Text Search Engines and Their Security Analysis"
4. Radu Gheorghe, Matthew Lee Hinman, and Roy Russo - „Elasticsearch in action" – November 2015 [Електронний ресурс] https://www.manning.com/books/elasticsearch-in-action
5. Руководство по Spring. Контейнеры IoC [Електронний ресурс] травень 2021 https://proselyte.net/tutorials/spring-tutorial-full-version/ioc-containers/
6. Craig Walls „Spring in Action, 6th edition" [Електронний ресурс] https://www.manning.com/books/spring-in-action-sixth-edition#toc
7. Anirban Chatterjee "Difference Between @ComponentScan and @EnableAutoConfiguration in Spring Boot" March 29, 2021 [Електронний ресурс] https://www.baeldung.com/spring-componentscan-vs-enableautoconfiguration
8. Документація SpringBoot Annotation Type SpringBootApplication https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/autoconfigure/SpringBootApplication.html
9. Spring Data Elasticsearch - Reference Documentation [Електронний ресурс] https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#preface