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

Розробка сервісу для пошуку та збереження документів для навчального закладу
Курсова робота за спеціальністю „Системний Аналіз” 124

Керівник курсової роботи
доктор техн. наук, доц. Глибовець А.М.

_____
(підпис)

“____” _____ 2021 р.

Виконав
студент 1 курсу магістерської програми
факультету інформатики
Картавий М. О.

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

ЗАТВЕРДЖУЮ
Декан факультету інформатики,
доктор техн. наук, доц. Глибовець А.М.

_____
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Картавому Миколі Олексійовичу факультету інформатики 1 курсу навчання магістерської програми

ТЕМА Розробка сервісу для пошуку та збереження документів для навчального закладу

Вихідні дані:
Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Календарний план
Анотація
Вступ
РОЗДІЛ 1: Дослідження технологій для ядра системи
РОЗДІЛ 2: Основи та особливості пошукового двигуна від Open Distro
РОЗДІЛ 3: Розробка сервісу
РОЗДІЛ 4: Тестування сервісу
Висновки
Список використаної літератури
Додатки

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

Тема: Розробка сервісу для пошуку та збереження документів для навчального закладу

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	07.11.2020	
2.	Пошук тематичної літератури	14.11.2020	
3.	Ознайомлення з тематичними матеріалами та побудова структури практичної та теоретичної частин курсової роботи	29.11.2020	
4.	Написання вступу та змісту роботи	04.01.2021	
5.	Ознайомлення з технологями для ядра системи	08.01.2021	
6.	Ознайомлення зі способами виділення ознак з аудіо сигналу та написання другого розділу	12.01.2021	
7.	Основи та особливості пошукового двигуна від Open Distro	15.02.2021	
7.	Розробка сервісу	22.02.2021	
8.	Тестування сервісу	18.03.2021	
9.	Оформлення роботи відповідно до вимог написання курсової роботи.	29.03.2021	
10.	Створення презентації та написання доповіді для захисту курсової роботи.	2021	
11.	Узгодження попередньої версії роботи з керівником	2021	
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	2021	
13.	Захист курсової роботи	2021	

Студент Картавий М.О.

Керівник Глибовець А.М.

“_____”_____

Зміст

Використані терміни в роботі

Документ – одиниця текстової інформації.
Індексація – процес додавання інформації для подальшого повнотекстового пошуку.
Інстанс – екземпляр об’єкту.
Плагін – розширення основного функціоналу у вигляді додаткового застосунку.
Обернений індекс (в повнотекстовому пошуку) – індекс, котрий використовується для повнотекстового пошуку, слово виступає в ролі індексу, на який посилаються інші об’єкти.
TF-IDF - статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів чи корпусу. Вага (значимість) слова пропорційна кількості вживань цього слова у документі, і обернено пропорційна частоті вживання слова у інших документах колекції.[9]
VSM(Vector Space Model) - в інформаційному пошуку алгебраїчне представлення колекції документів векторами одного спільного для всієї колекції векторного простору.[10]

Анотація до курсової роботи

Мета дослідження: розробити систему для збереження великої кількості великої кількості документів та їх швидкого пошуку, редагування та контролю доступу для перегляду, також інструмент для адміністратора, що дозволить аналізувати та створювати звітність на основі даних.

Джерела для дослідження: використані іноземні джерела – наукова література та статті. Більш детально можна ознайомитися в розділі «Список використаних джерел».

Що було зроблено: ми розглянули технології, що підуть в основу системи для реалізації нашої задачі, обрали ту, котра краще задовольняє наші потреби, протестували швидкість роботи та підбили підсумки.

Дані для тестування системи: Дані, котрі використовуються для тестування системи – набір різних типів документів у кількості 8000 (вісім тисяч).

Вступ

Протягом останніх років відбувається масштабний перехід документів паперової печаті в цифровий формат. Так відбувається, оскільки цифрові документи мають велику кількість переваг: швидкий пошук інформації серед усіх документів, зменшення витрат на підтримку та збереження документів, легкість в менеджменті, цифрові документи не потребують догляду під час зберігання, тощо.

Чи використовується цифровий документообіг в великих компаніях – неодмінно, замість того, щоб зберігати велику кількість паперів, можна зберегти на цифровому носії, зробити бекап для запобігання втрати, швидко знайти документ/інформацію в ньому, відредагувати частину документу – це все простіше робити з цифровим документом, ніж паперовим. Наразі й наш університет – НаУКМА переходить на платформу цифрового документообігу.

Вимоги до системи

1) Швидкий пошук документів за різними характеристиками
2) Зберігання
3) Видалення документів
4) Редагування документів
5) Підтримка різних форматів документу
6) Ведення історії редагувань документів
7) Контроль доступу перегляду/створення/редагування/видалення документа
8) Інструмент для аналізу та створення звітності на основі даних в системі

Наша система повинна в першу чергу проводити швидкий пошук по документам, оскільки пошук – основна причина переходу на цифровий документообіг. Зберігання та видалення документа – рідше використовуються, тому пріоритет мають нижчий, а отже можуть бути не такими швидкими. Редагування документа проходить ще рідше, тому можна не акцентувати на цьому увагу. Підтримка різних форматів документу – документ може бути зроблений в різних програмах для створення текстових файлів, тому наша система повинна підтримувати усі основні формати текстових файлів. Ведення історії – додаткова функція, котра дозволяє слідкувати за змінами в документі. Контроль доступу перегляду повинен встановлювати – хто може бачити цей документ, або хто може створювати/видаляти/редагувати таку групу документів.

Розділ 1

Дослідження технологій для ядра системи

1.1 Технології для швидкого повнотекстового пошуку

Нас цікавила технологія, котра дозволяє оперувати серед великої кількості текстових інформації, на цю потребу підходило кілька сервісів.

1) Elasticsearch – пошуковий двигун з підтримкою JSON REST API, має багато плагінів для розширення функціоналу та додаткових сервісів, зараз є платним.
2) Open Distro – розповсюдження Elasticsearch під ліцензією Apache 2.0 з додатковими функціями.
3) Solr – пошукова платформа на базі Apache Lucene. [7]
4) ArangoDB – альтернатива Elasticsearch, з дещо розширеними функціями.[6]

Ми побудуємо таблицю з критеріїв та їх відповідності до пошукової системи. З критеріїв нам важливі: пошук документів за змістом, підтримка ОС, підтримка поширених форматів текстових файлів (.docx, .doc,.pdf,.txt,.xls, .xlsx, .ppt), безкоштовність, плагіни для розширення функціоналу та по можливості вбудований інструмент для створення звітності на основі даних в системі.

Таблиця дозволить наглядно порівняти системи та обрати найбільше актуальну для нас. Нам дуже важливо, щоб система могла автоматично опрацьовувати документи та була безкоштовною, оскільки критерій швидкого пошуку у всіх є однаковим – всі системи проводять пошук документу в режимі реального часу.

Назва технології/Функціонал	Elasticsearch	Open Distro	Solr	ArangoDB
Підтримка ОС	Будь яке середовище з Java	Windows, Linux або середовище з Docker	Будь яке середовище з Java	Linux
Пошук документів	В режимі реального часу	В режимі реального часу	В режимі реального часу	В режимі реального часу
Підтримка текстових документів (.docx, .doc,.pdf,.txt,.xls, .xlsx, .ppt)	З плагіном для парсингу документів	З плагіном для парсингу документів	Немає	Немає
Безкоштовний для застосування	Ні	Так	Так	Ні
Плагіни	Так	Всі, що на Elasticsearch	Ні	Ні
Інструмент для аналізу системи	Kibana	Kibana	Немає	Немає

Отже, можна зробити висновок, що найбільше для розробки нашої системи підходить Open Distro, обираючи Open Distro ми одразу вирішуємо проблему з парсингом документів поширених форматів, інструментом для аналізу системи та безкоштовністю.

Розділ 2

Основи та особливості пошукового двигуна від Open Distro.

2.1 Схема даних

Open Distro – це розповсюдження Elasticsearch, то необхідно розібрати як працює Elasticsearch. Elasticsearch – це двигун з неструктурованою схемою даних, тому про зовнішні ключі можна забути. Оскільки пошук повинен бути швидким, то інформацію краще зберігати всю (що стосується певного документа) в одному місці. Тобто замість типової структури в базі даних (рис.1) ми маємо все в одному об'єкті (рис.2). [1]

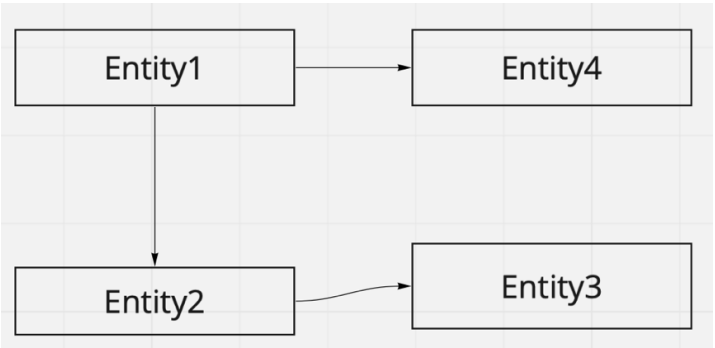


Рисунок 1 – Зв'язки між сутностями в типовій БД

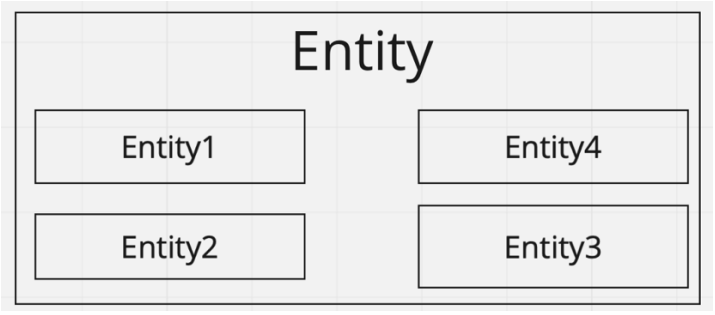


Рисунок 2 – Приклад вмісту сутності в Elasticsearch

2.2 Пошук

Для швидкого пошуку в базах даних використовуються індекси – позначається об’єкт індексом, на котрий потім інший об’єкт може посилатися. В нашому випадку необхідно проводити пошук за документами, тож для цього використовується повнотекстовий пошук - за всім змістом документу, що є досить не швидко на великій кількості документів, тоді виникає питання як зробити цей пошук швидким? Треба індексувати документ – можна зробити документ як індекс і під’єднати всі слова до цього документу (рис. 3), тоді ми будемо знати всі слова, котрі зустрічаються в документі – об’єкт швидко індексується, проте це не вирішує проблему швидкого пошуку. Можна індексувати зміст документу – кожне слово в документі позначити як індекс і під’єднати документи до слова, котре зустрічається в цих документах (рис. 4). Тоді обравши одне слово – ми можемо знайти всі документи, котрі мають це слово і це швидко. Процес парсингу та індексування слів займе більшу кількість часу (залежить від об’єму документу), та це не важливо, оскільки важливий швидкий пошук, а не повільна індексація. [1]

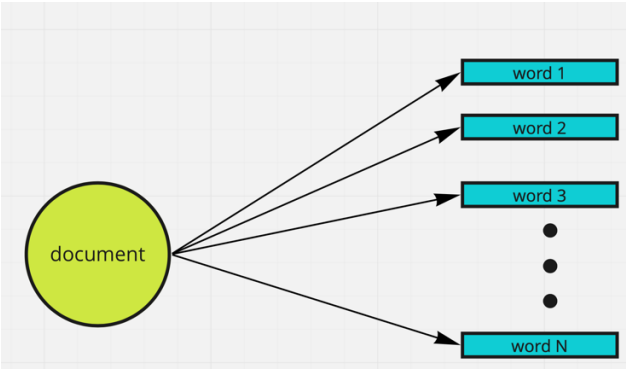


Рисунок 3 – Документ виступає в ролі індексу

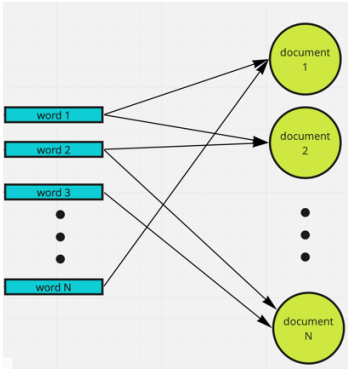


Рисунок 4 – Слово виступає в ролі індексу

Саме цей спосіб індексування і використовує Elasticsearch, точніше Elasticsearch використовує бібліотеку Apache Lucene, котра використовує такий спосіб індексування - обернений. [1]

2.3 Масштабування

Існує два основні типи масштабування: горизонтальне та вертикальне. Горизонтальне масштабування – збільшення кількості машин для вирішення задачі, складний, з мережевими затримками, але надійний спосіб. Вертикальне масштабування – збільшення потужності однієї машини, швидко, без мережевих затримок, проте не надійно та має обмеженість. Як правило, компанії або установи, котрі вирішують перейти на цифровий документообіг, роблять це із-за дуже великої кількості документів, тому масштабування є досить важливим фактором в документообігу – документів може бути тисячі, а може й десятки тисяч. Для того, щоб зробити горизонтальне масштабування, необхідно розбити весь індекс документів на частини – shard (шард). Саме шарди виконують пошук та збереження документів, а індекс має набір шардів (мінімум 1). Оскільки індекс складається з багатьох шардів, то шукати по всім шардам одного індексу – складна задача, для вирішення цього можна робити індекс для кожного типу документів (курсова робота, дипломні, накази, відгуки і т. д.). Тоді можна розмістити індекс під кожен тип документів по різним машинами, а формувати запит, направляти та об’єднувати результати з різних індексів будуть координуючі вузли. Це є досить гарною практикою для того, щоб звузити пошук конкретного документу. [1]

2.4 Структура Elasticsearch

На рисунку 5 зображено схематично структуру Elasticsearch.

Cluster – вузол системи, котрий координує запити до shard-ів. Він виконує функцію обробки запиту, передавання запиту до shard-ів та збору відповіді від них.

Shard – шматок Index-у, вузол з даними, котрий дозволяє розбити інформацію за певними критеріями, наприклад, за частотою пошуку – можна зберігати дані, котрі рідко шукаються в одному shard-і, а дані, котрі нові чи найчастіше використовуються в іншому shard-і, це дозволяє ще пришвидшити пошук актуальної інформації.

Data-node – вузол з набору shard-ів, котрий дозволяє робити забезпечити безпеку файлів – не втратити їх. Так, наприклад, shard з однієї data-node має містити свою копію на іншій data-node, тоді це дозволить в разі втрати швидко дістати інформацію з іншої data-node, поки відбуваються роботи по відновленню. [1]

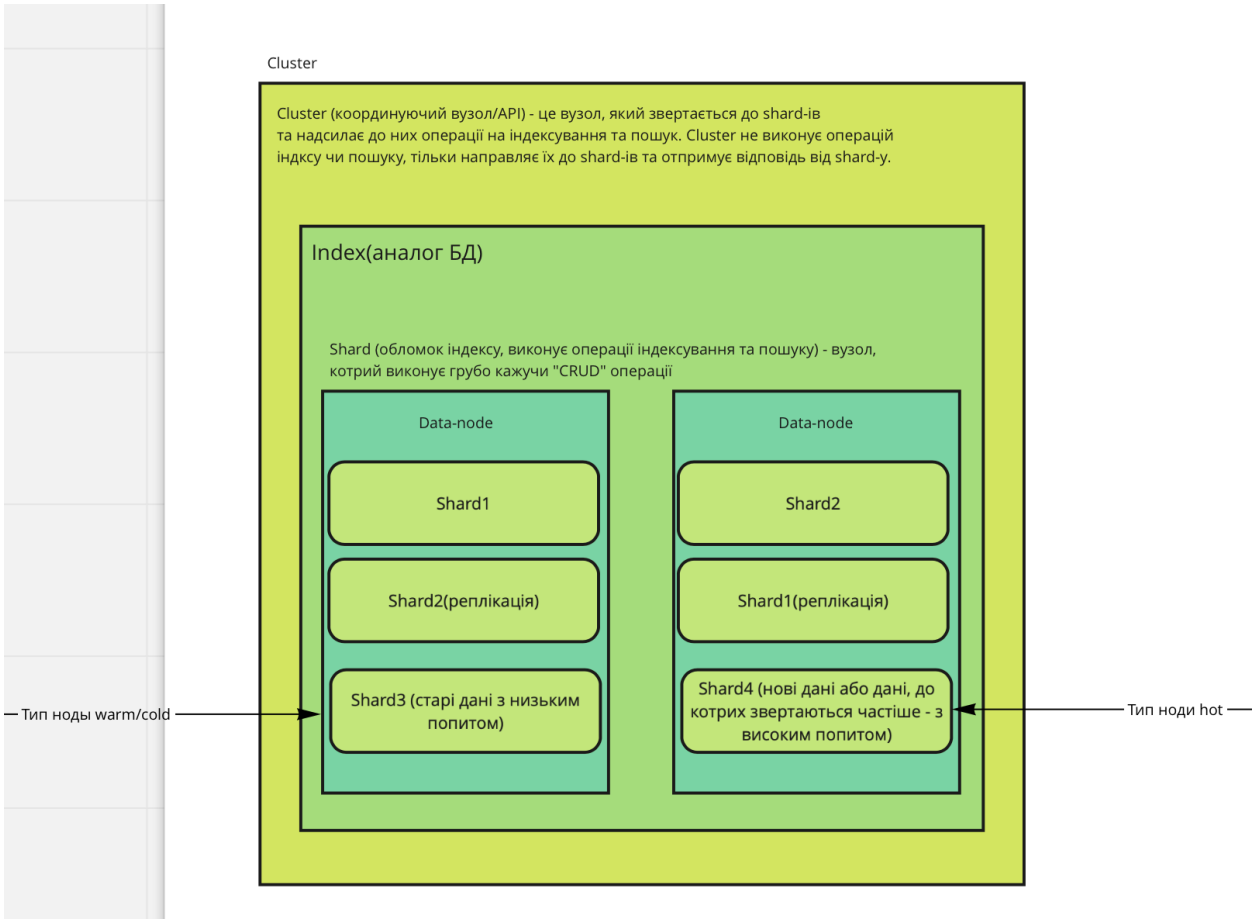


Рисунок 5 – Структура Elasticsearch

Розділ 3

Розробка сервісу

3.1 Вибір фреймворку для основи сервісу

Оскільки Elasticsearch написаний на Java, більшість додаткових бібліотек теж на Java, тому вибір мови пав на Java. Серед популярних фреймворків для розробки серверних Java застосувань існують: Grails, Play, Spring. Вибір пав на Spring, оскільки він має більше додаткових бібліотек та й частіше використовується в ентерпрайзі.

Отже, стек вийшов такий:

- Java 14 + Spring Boot 2
- Maven
- Open Distro
- PostgreSQL
- Flyway
- Jupiter 5
- Spring Data
- Spring Security
- Spring MVC

3.2 Опис структури додатку

Сервіс документообігу повинен вбудовуватися в іншу систему, тому було прийняте рішення – розбити додаток на модулі та відповідно відокремити: спосіб комунікації з нашим додатком, моделі даних та логіку застосунку. Отже, ми можемо кожен модуль скомпілювати в окремий додаток та потім використовувати як бібліотеку/окремий застосунок. Схема модулів (рис. 6), на схемі позначено:

- document-management-api-client
- document-management-model
- document-management-service

Модуль document-management-model не є самостійним та використовується в інших двох модулях. Модуль document-management-api-client використовується як бібліотека, що визначає інтерфейс для зв'язку з модулем document-management-service, котрий є основною частиною застосунку.

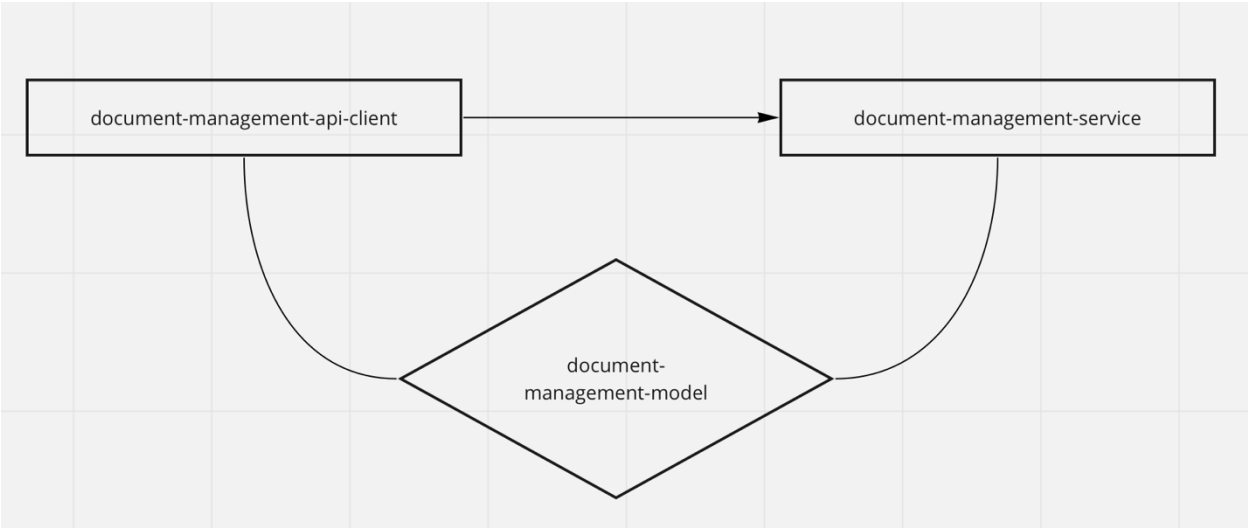


Рисунок 6 – Взаємодія модулів у додатку

3.3 Побудова архітектури бази даних

Наш документообіг є мікросервісом, оскільки, виконує тільки управління документами та може вбудовуватися в іншу навчальну систему, тому необхідно було побудувати базу даних тільки для частини, котра стосується управлінням доступу документів. Для вирішення задачі з доступами ми використали ролі та дозволи. Ми побудували невеличку базу з трьох таблиць: role, permission, role_permission. Така схема бази даних дозволить легко оперувати доступами для конкретної ролі, зв'язок між role та permission – багато до багатьох.

Схема виглядає так (рис. 7) :

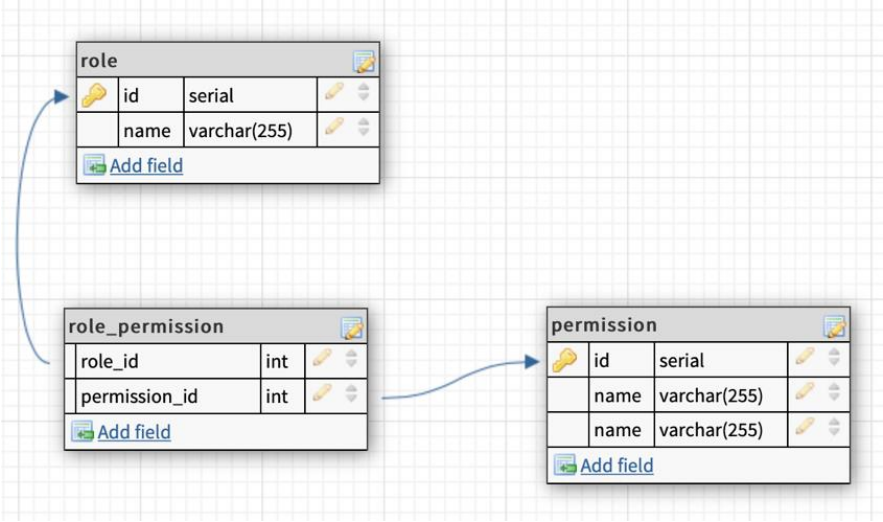


Рисунок 7 – Схема бази даних для керування доступами в застосунку

SQL скрипт:

```
CREATE TABLE IF NOT EXISTS role(
   id   SERIAL PRIMARY KEY,
   name VARCHAR(45) NOT NULL UNIQUE
);
CREATE TABLE IF NOT EXISTS permissions(
   id   SERIAL PRIMARY KEY,
   name VARCHAR(45) NOT NULL UNIQUE
);
CREATE TABLE IF NOT EXISTS role_permissions(
   role_id      INT NOT NULL REFERENCES role (id) ON DELETE NO ACTION,
   permission_id INT NOT NULL REFERENCES permissions (id) ON DELETE
NO ACTION,
   PRIMARY KEY (role_id, permission_id)
);
```

3.4 Побудова архітектури документів

Наш застосунок в першу чергу розроблявся для покращення цифрової систему нашого університету – НаУКМА, тому вищий навчальний заклад виділяє 9 основних типів документів:

- Положення
- Накази по університету
- Накази по факультету
- Дипломні роботи
- Курсові роботи
- Відгуки на дипломні роботи
- Силабуси
- Навчальні плани
- Інші документи

Отже, можна відокремити всі ці типи документів в окремі API:

- act-api
- university-order-api
- faculty-order-api
- graduate-work-api
- course-work-api
- review-graduate-work-api
- syllabus-api
- education-plan-api
- other-docs-api

це дозволить структурувати документи та відокремити доступи по ролям.

Спираючись на типи документів ми обрали 5 ролей: UNIVERSITY_STAFF, FACULTY_STAFF, STUDENT, TEACHER та HELPER. Також ми продумали доступи до конкретних API, що дозволило побудувати прозору схему роботи кожної ролі. Нижче ми опишемо схему API (рис. 8), кожне з яких позначимо окремим кольором, щоб потім не було плутанини зі стрілками, та навпроти кожної ролі позначимо кружечками відповідного кольору (рис. 9 – створювати документ відповідного типу, рис. 10 – читати документ відповідного типу, рис. 11 – видаляти документ відповідного типу, рис. 12 – оновлювати документ відповідного типу) які документи ця роль може створювати/читати/оновлювати/видаляти.

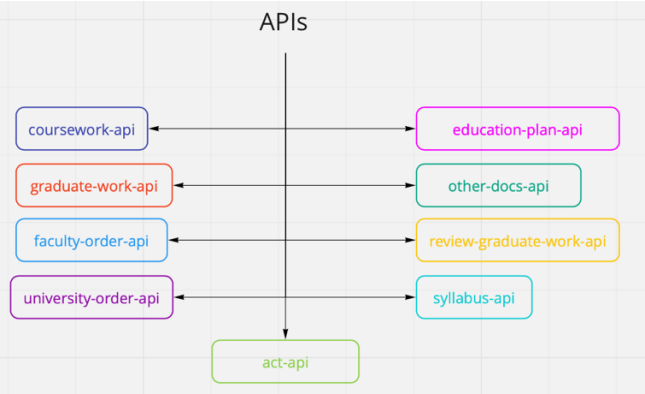


Рисунок 8 – Схема API

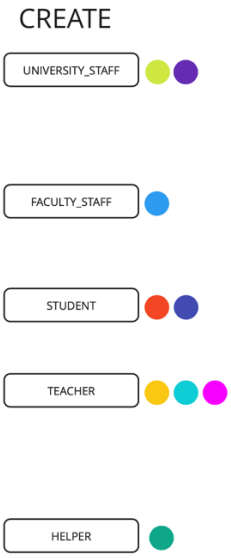


Рисунок 9 – відповідності ролей та можливості створювати документ через певний API

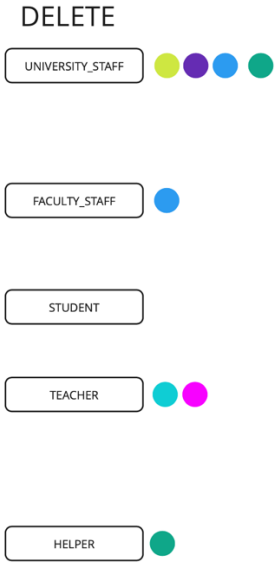


Рисунок 11 - відповідності ролей та можливості видаляти документ через певний API

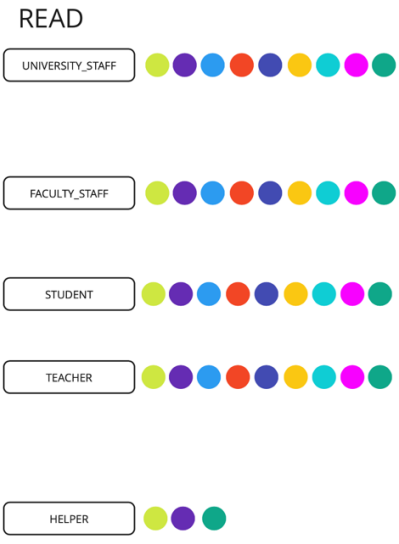


Рисунок 10 - відповідності ролей та можливості читати документ через певний API

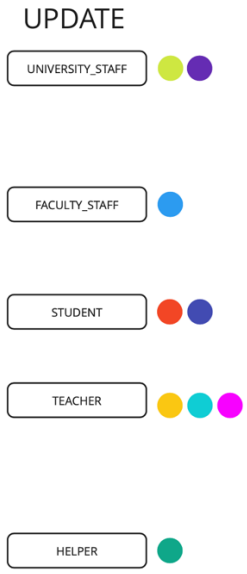


Рисунок 12 - відповідності ролей та можливості оновлювати документ через певний API

3.5 Встановлення Open Distro

Для того, щоб встановити Open Distro ми використовували Docker.

1) Встановлюємо Docker з офіційного сайту https://www.docker.com/get-started
2) Створюємо конфігураційний файл для Docker з налаштуваннями для Elasticsearch або перейдіть за посиланням https://opendistro.github.io/for-elasticsearch/downloads.html#try та завантажте “docker-compose.yml”
3) Запускаємо команду “docker-compose up” у папці з файлом налаштувань “docker-compose.yml”
4) Open Distro for Elasticsearch налаштований [2]

Тепер Elasticsearch за замовчуванням знаходиться на https://localhost:9200, авторизація за допомогою логіну - admin та паролю – admin, а Kibana – інструмент для аналізу системи, знаходиться на https://localhost:5601

3.6 Налаштування плагіну для автоматичного парсингу текстових файлів з розширенням (.pdf, .doc, .docx, .xls, .xlsx, .ppt)

Для того, щоб зчитувати зміст файлів з розширенням (.pdf, .doc, .docx, .xls, .xlsx, .ppt) автоматично, необхідно встановити плагін «ingest-attachment» .

1) Необхідно перейти в папку bin, де розташований Open Distro for Elasticsearch
2) Запустити команду «sudo bin/elasticsearch-plugin install ingest-attachment» через термінал або командну стрічку. Для Docker необхідно запустити Docker, запустити інстанс, запустити CLI для кожної з нод та запустити команду «bin/elasticsearch-plugin install ingest-attachment».
3) Після встановлення плагіну можна його використовувати для автоматичного парсингу змісту файлів з розширенням (.pdf, .doc, .docx, .xls, .xlsx, .ppt)

3.7 Налаштування конфігурації додатку для роботи з Open Distro for Elasticseacrh

ElasticSearchConfig.java – клас конфігурації застосунку Spring Boot для роботи з Elasticsearch. [8]

@Slf4j – анотація з бібліотеки Lombok для автоматичного створення логера.

@Configuration – анотація, котра вказує на те, що даний клас є конфігураційним у додатку.

@EnableElasticsearchRepositories – анотація, що вказує на автоматичне сканування репозиторіїв для документів. [8]

@AllArgsConstructor - анотація з бібліотеки Lombok для автоматичного генерування конструктору за всіма полями класу.

Поле «elasticVariables» - об’єкст, що містить константи для конфігурації з’єднання з Ealsticsearch.

Налаштування SSLContext - оскільки Open Distro for Elasticseacrh розрахований на застосування у хмарі, то має шифрування протоколу http, зараз ми пропускаємо всі сертифікати, але в майбутньому необхідно згенерувати сертифікат та використовувати його.

RestHighLevelClient - клієнт для «спілкування» з Elasticseacrh. [8]

ElasticsearchOperations - клас для запитів-операцій з документом до Elasticsearch (індексувати файл, дістати, інші).

Метод «createPipeline» викликається одразу після закінчення роботи конструктора танадсилає конфігуруючий запит, що буде застосовувати плагін ingest-attachment до певного поля кожного документу.

На наступній сторінці представлений відформатований код.

```
@Slf4j
@Configuration
@EnableElasticsearchRepositories
@AllArgsConstructor
public class ElasticSearchConfig {

   private ElasticSearchVariables elasticVariables;

   @Bean
   public RestHighLevelClient client() throws NoSuchAlgorithmException, KeyManagementException,
KeyStoreException {
      SSLContext sslContext = new SSLContextBuilder()
            .loadTrustMaterial(null, (x509CertChain, authType) -> true)
            .build();
      ClientConfiguration clientConfiguration = ClientConfiguration.builder()
            .connectedTo(elasticVariables.getHost())
            .usingSsl(sslContext, (s, sslSession) -> {
               log.info(s);
               return true;
            })
            .withBasicAuth(elasticVariables.getUser(), elasticVariables.getPassword())
            .build();

      return RestClients.create(clientConfiguration).rest();
   }

   @Bean
   public ElasticsearchOperations elasticsearchTemplate() throws KeyManagementException,
NoSuchAlgorithmException, KeyStoreException {
      log.info("Create ElasticSearch client");
      return new ElasticsearchRestTemplate(client());
   }

   @PostConstruct
   private void createPipeline() throws IOException, KeyManagementException, NoSuchAlgorithmException,
KeyStoreException {
      Pipeline pipeline = new Pipeline.PipelineBuilder(1)
            .setDescription("attachment-processor")
            .addAttachmentProcessor(new Processor()
                  .setAttachment(new AttachmentProcessor()
                        .setField(elasticVariables.getPipelineAttachmentField())
                        .setIgnoreMissing(true)
                        .setIndexedChars(-1L)))
            .build();
      PutPipelineRequest request = Requester.buildRequest(elasticVariables.getPipelineAttachmentId(),
pipeline);
      AcknowledgedResponse response = client().ingest().putPipeline(request, RequestOptions.DEFAULT);
      log.info("Pipeline isAcknowledge: " + response.isAcknowledged());
   }

}
```

[8]

3.8 Схема класу-обгортки документу

Анотації: @Document, @Field – є необхідними для коректної роботи Spring Data з Elasticsearch, також вони є аналогом до Hibernate анотацій:@Table+@Entity та @Column відповідно. Тому для операцій видалення та операцій читання необхідно визначити клас-обгортку для JSON даних, котрі зберігаються Elasticsearch. [8]

@Document – анотація, що позначає до якого індексу документів клас-обгортка належить.

@Field - анотація, що позначає поле якого типу повинно бути для зберігання в індексі. Має кілька аргументів: type та format, type- обов'язковий.

```
@Document(indexName = ActDoc.INDEX)
class ActDoc extends DocumentDoc {

   public static final String INDEX = "act_index_0";

   @Field(type = FieldType.Text)
   protected String title;

   @Field(type = FieldType.Integer)
   private Integer version;

   @Field(type = FieldType.Date, format = DateFormat.basic_date_time)
   private LocalDateTime approvalDateTime;
}
```

[8]

3.9 Операції з документами

Процес роботи з документами схожий до роботи зі звичайною базою, також необхідно документ індексувати, оновлювати, видаляти та читати, проте використання Spring Data в парі з плагіном для автоматичного парсингу змісту документу не є можливим, тому було поєднано використання Spring Data та запити з бібліотеки org.elasticsearch.client, ми реалізували свій «круд-сервіс» для роботи з документами. Для індексування та читання документів використовуються такі класи з бібліотеки:

- IndexRequest – запит для додавання документу в індекс
- CriteriaQuery – клас для побудови умов для фільтрування даних

Для видалення використовувалися Spring Data. Для оновлення використовувалися видалення документу та індексування по-новому, оскільки Elasticsearch не підтримує функцію оновлення проіндексованого документу.

Нижче представлено схематичний процес індексування (рис. 13), проте ця схема є аналогічною й для інших запитів:

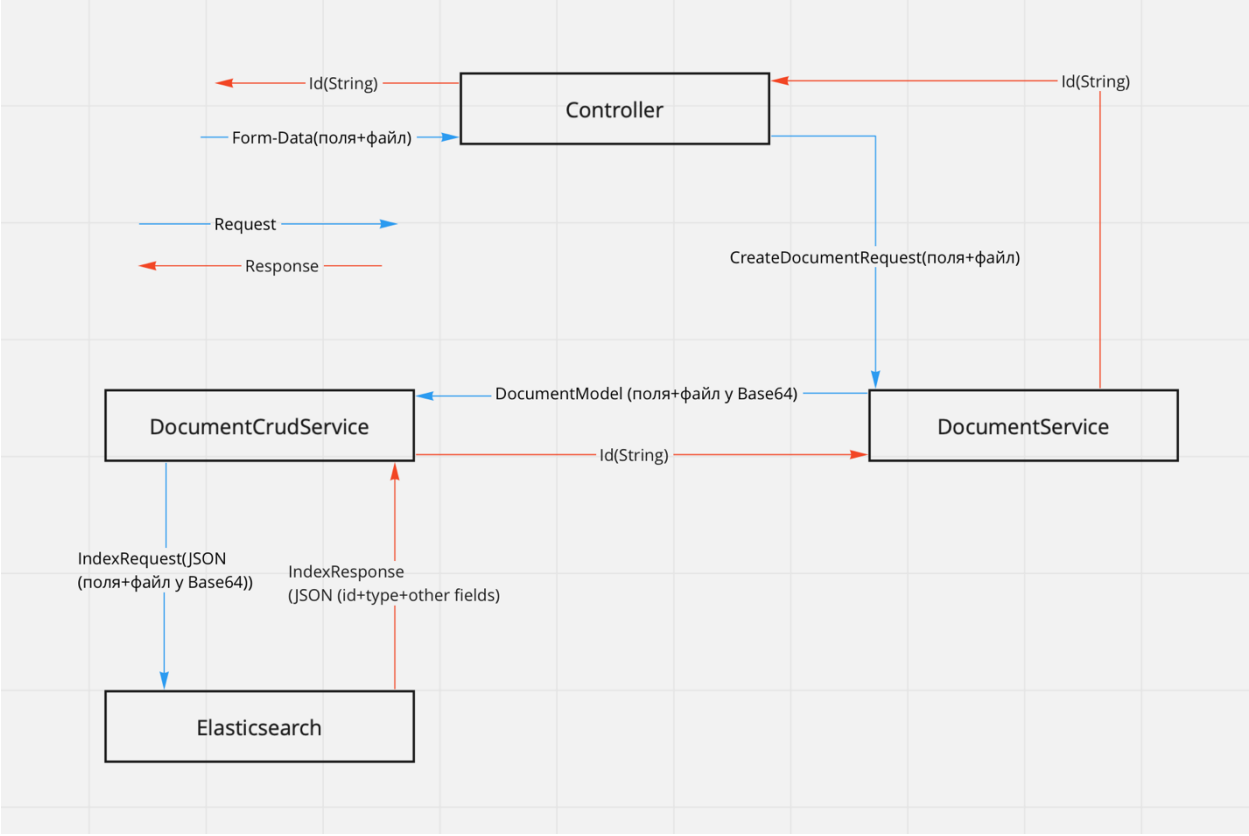


Рисунок 13 – схематичний процес індексування документу, синім кольором позначено запит від client, червоним – відповідь від сервісу

Ми можемо бачити як запит від клієнту потрапляє до контролеру, після чого передається в сервіс бізнес-логіки, сервіс бізнес-логіки передає запит до круд-сервісу, а круд-сервіс вже працює з Elasticsearch сервісом. Отже, таким чином ми відокремили логіку процесів в додатку та зробили її прозорою.

Розділ 4

Тестування сервісу

Анотація до розділу

Elasticsearch використовує бібліотеку Apache Lucene, яка в свою чергу використовує, VSM(Vector Space Model) в поєднанні з TF-IDF, TF-IDF використовується для того, щоб співставити кожному терміну набір документів за релевантністю. Отже, ми маємо набір документів, кожен з яких представлений у вигляді вектора термінів та запит у вигляді вектора зі слів. Виходить, що відбувається пошук відповідних документів за терміном, після чого відбувається сортування за релевантністю – вираховується TF-IDF для запиту та ставиться у відповідність набір документів за спаданням ваги. Тому складність пошуку залежить від кількості документів, кількості слів в документах та кількості слів у запиті.

4.1 Підготовка даних для тестування

Для тестування швидкості пошуку за змістом, було згенеровано вибірку з 8000 (восьми тисяч) документів. Це документи типу .docx та кожен з яких кількістю слів близько 200.

4.2 Етапи тестування

Проводити тестування необхідно на різних кількостях даних збережених одночасно, для того, щоб потім побачити, тому ми розбили тестування на етапи:

- 2000 документів одночасно збережених та пошук 2000 з них
- 4000 документів одночасно збережених та пошук 2000 з них
- 6000 документів одночасно збережених та пошук 2000 з них
- 8000 документів одночасно збережених та пошук 2000 з них

4.3 Спосіб тестування

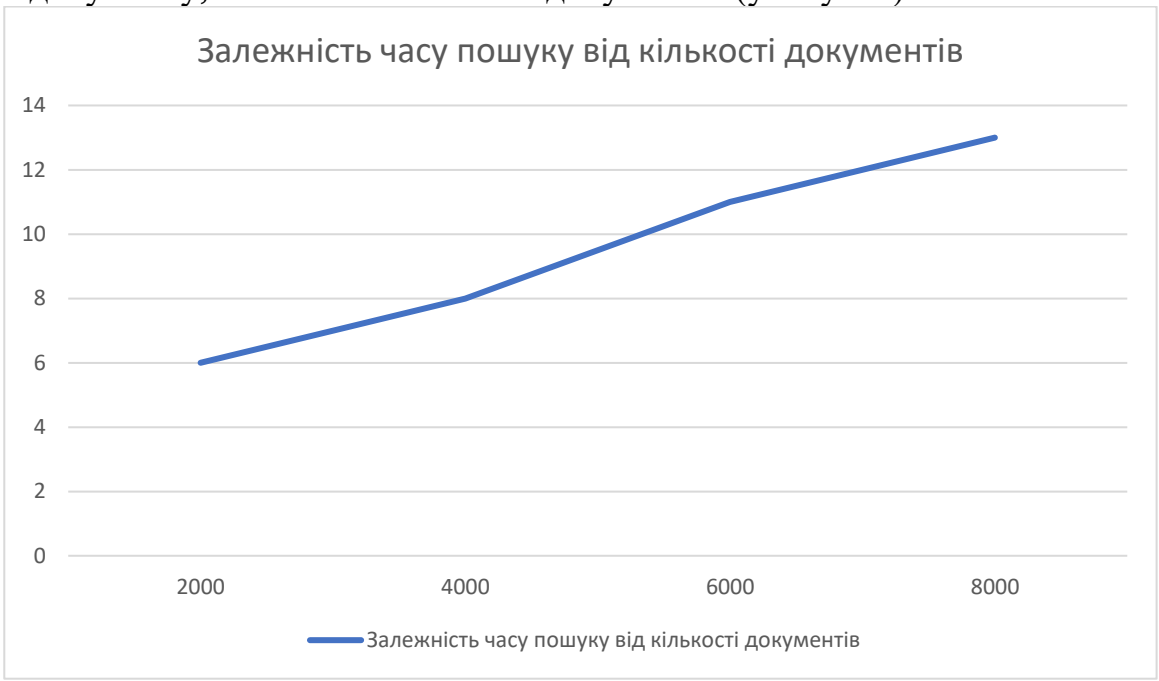
Тестування пошуку за змістом відбувалося за одним словом з документу, та якщо результат пошуку повертав відповідь зі списком документів, в якому містилися документи, котрі мають це слово – тест проводиться вірно. Також для усереднення часу та запобігання можливих сторонніх факторів (процеси, котрі запустила система тестуючої машини) кожен з етапів проводився 100 (сто) разів. Після чого вираховується середній час на пошук кожного з документів.

4.4 Конфігурація машини для тестування

OS: MacOS Catalina 10.15.7 (19H2)
CPU: Intel Core I7 8750H
RAM: 16 GB DDR4 2400 Mhz
SSD: 250 GB

4.5 Результати тестів

Нижче показаний графік залежності часу пошуку від кількості одночасно проіндексованих документів. По осі У – час (у мс) для пошуку одного документу, по осі Х – кількість документів (у штуках).



Отже, можна зробити висновок, що час затрачений на пошук одного документа не є рівним для різної кількості одночасно проіндексованих документів, як і сподівалося, можна бачити яскраву залежність часу від кількості проіндексованих документів.

Висновок

Отже, за завершенням даної роботи було розроблено додаток для цифрового документообігу, проведено тестування швидкості роботи додатку та стабільності швидкості роботи з ростом кількісті документів. Дана курсова робота дозволила дізнатися більш детально про технологію Elasticsearch та використати її на практиці з метою покращення цифрової системи навчання. Порівняти аналоги технології повнотекстового пошуку та обрати кращу під нашу задачу, вдосконалити навички проектування архітектури застосунків. Сподіваюся, що дана робота допоможе системі цифрового навчання та буде приносити користь університету.

Список використаних джерел

1) https://habr.com/ru/post/489924/

2) https://opendistro.github.io/for-elasticsearch/downloads.html#try

3) https://habr.com/ru/post/280488/

4) https://opendistro.github.io/for-elasticsearch/

5) https://www.elastic.co

6) https://www.arangodb.com

7) https://solr.apache.org

8) https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/#reference

9) https://uk.wikipedia.org/wiki/TF-IDF

10) https://uk.wikipedia.org/wiki/Векторна_модель