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

СИСТЕМА ЗБЕРІГАННЯ ТА ПОШУКУ ДИПЛОМНИХ ТА КУРСОВИХ РОБІТ

Текстова частина до курсової роботи
за спеціальністю „Прикладна математика” 6.040301

Керівник курсової роботи:
д.т.н., доцент, Глибовець А. М.
_____/підпис/
“_____” _____ 2020 р.

Виконав студент:
Добровольський І. С.
_____ /підпис/
“_____” _____ 2020 р.

Київ 2020

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)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Добровольському Ігорю Сергійовичу факультету інформатики 3-го курсу
ТЕМА Система зберігання та пошуку дипломних та курсових робіт
Вихідні дані:

- Застосунок – система зберігання та пошуку дипломних та курсових робіт

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
Розділ 5: Опис системи, особливості розгортки та налаштування
Розділ 6: Використання системи
Висновки
Список використаної літератури

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

Тема: Система зберігання та пошуку дипломних та курсових робіт

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи	10.11.2019	
2.	Огляд технічної літератури за темою роботи	30.11.2019	
3.	Встановлення необхідного програмного забезпечення	02.12.2019	
4.	Реалізація серверної частини застосунку	07.01.2020	
5.	Реалізація UI частини застосунку	14.01.2020	
7.	Реалізація автентифікації та авторизації в застосунку	20.01.2020	
8.	Написання основної частини курсової роботи	06.05.2020	
11.	Перегляд змісту роботи з керівником	08.05.2020	
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	09.05.2020	
13.	Створення слайдів для доповіді та написання доповіді	10.05.2020	
14.	Захист роботи		

Студент Добровольський І. С.

Керівник Глибовець А. М.

“_____”_____

Зміст

АНОТАЦІЯ

У процесі написання курсової роботи було створено систему для зберігання та пошуку дипломних та курсових робіт. Було розглянуто різні пошукові двигуни та обрано Elasticsearch. Для розробки серверної частини було використано Spring Framework. Для розробки веб-сторінки було використано фреймворк Angular. Для реалізації автентифікації та авторизації було використано Azure Active Directory.

У курсовій роботі міститься 22 рисунки.

Вступ

Щороку студенти здають свої дипломні та курсові роботи. Цей процес займає досить багато часу і є незручним як для самого студента, так і для наукового керівника: курсову роботу необхідно показати рецензенту та науковому керівнику, керівник має затвердити роботу та написати відгук, рецензію та відгук потрібно додати до роботи тощо. Більш того, з'являється необхідність здійснювати пошук по всім роботам.

Наразі всі роботи студентів університету зберігаються в каталозі на персональному комп'ютері. Таким чином, роботи не є доступними через Інтернет. Через відсутність веб-застосунка всі необхідні маніпуляції з роботою, наприклад, передача її рецензенту, додавання рецензії чи пошук, виконуються вручну, що вимагає більше часу та збільшує вірогідність допущення помилки.

За для спрощення та автоматизації деяких етапів написання дипломних та курсових робіт, а також зручного зберігання та пошуку, за мету даної роботи було поставлено розробку веб-застосунка, який надаватиме сервіс для зберігання, перевірки та пошуку робіт.

Використання та подальша розробка даного веб-застосунка дозволить створити єдину платформу дипломних та курсових робіт університету, а в майбутньому й масштабувати до всіх вищих навчальних закладів країни.

РОЗДІЛ 1: Системи зберігання та пошуку

Система зберігання та пошуку – це система, що надає користувачу можливість зберігати певну інформацію та здійснювати пошук збереженої інформації за запитом. Основні вимоги до такої системи:

а) система має бути доступною для користувача;

б) система повинна гарантувати цілісність інформації;

в) збережена інформація має бути доступною лише для авторизованих користувачів.

РОЗДІЛ 2: Пошукові двигуни

2.1. Пошукові сервери на базі Apache Lucene

Пошуковий двигун надає можливість швидко здійснювати пошук на великих даних, індексуючи їх попередньо. Пошуковий сервер – сервер, що надає можливість працювати з пошуковим двигуном використовуючи протокол HTTP. Було розглянуто 2 пошукових сервери: Elasticsearch та Apache Solr. Обидва сервери написані мовою програмування Java, базуються на бібліотеці для повнотекстового пошуку Apache Lucene та легко інтегруються з Spring Framework.

2.2. Огляд Elasticsearch

Elasticsearch надає можливість здійснювати розподілений пошук та працювати з одним сервером, незалежно розділяючи дані (multi-tenancy), що надає можливість різним непов’язаним застосункам працювати з одним сервером [1].

Elasticsearch надає можливість здійснювати пошук за запитами багатьох видів. Можливі запити включають в себе:

а) булеві запити;

б) повнотекстові запити;

в) гео-запити;

г) запити за подібними документами;

ґ) запити за скриптами;

д) запити з використанням позицій.

2.3. Огляд Apache Solr

Apache Solr надає можливість розподіленого індексування, репліціювання, балансування навантаження та автоматичного відновлення інформації [2].
Apache Solr також є стійкою до відмов.
Apache Solr надає можливість здійснювати повнотекстовий пошук.

РОЗДІЛ 3: Предметна область

Для зберігання дипломних та курсових робіт потрібно зберігати назву, всі документи та інформацію про автора. Оскільки робота повинна пройти схвалення від керівника, автор повинен мати лише можливість створення запиту на збереження роботи. Перевіривши роботу, керівник має можливість

підтвердити запит, або скасувати його з відповідним коментарем. Отже, система має 2 ролі: звичайний користувач та адміністратор, який може переглядати, схвалювати та скасовувати запити.

РОЗДІЛ 4: Огляд стеку технологій

4.1. Огляд Spring

Spring – це сучасний фреймворк для створення бізнес-застосунків на мові Java [3]. Spring було використано для побудови веб-додатку з метою спрощення роботи з Elasticsearch, автентифікацією і авторизацією.

4.2. Огляд Angular

Angular – це фреймворк для розробки додатків, що складається з одної HTML сторінки, CSS та JavaScript [4]. Angular було використано для швидкої розробки front end частини з використанням таких готових компонентів, як поля для вводу, кнопки, таблиці та діалогові вікна.

РОЗДІЛ 5: Опис системи, особливості розгортки та налаштування

5.1. Перелік компонентів веб-застосунку

Веб-застосунок було розбито на 4 компоненти:

а) front end;

б) основна серверна частина;

в) компонент для роботи з автентифікацією і авторизацією;

г) компонент для роботи з Elasticsearch.

5.2. Діаграма класів

Діаграма всіх класів зображена на рисунку 5.1.

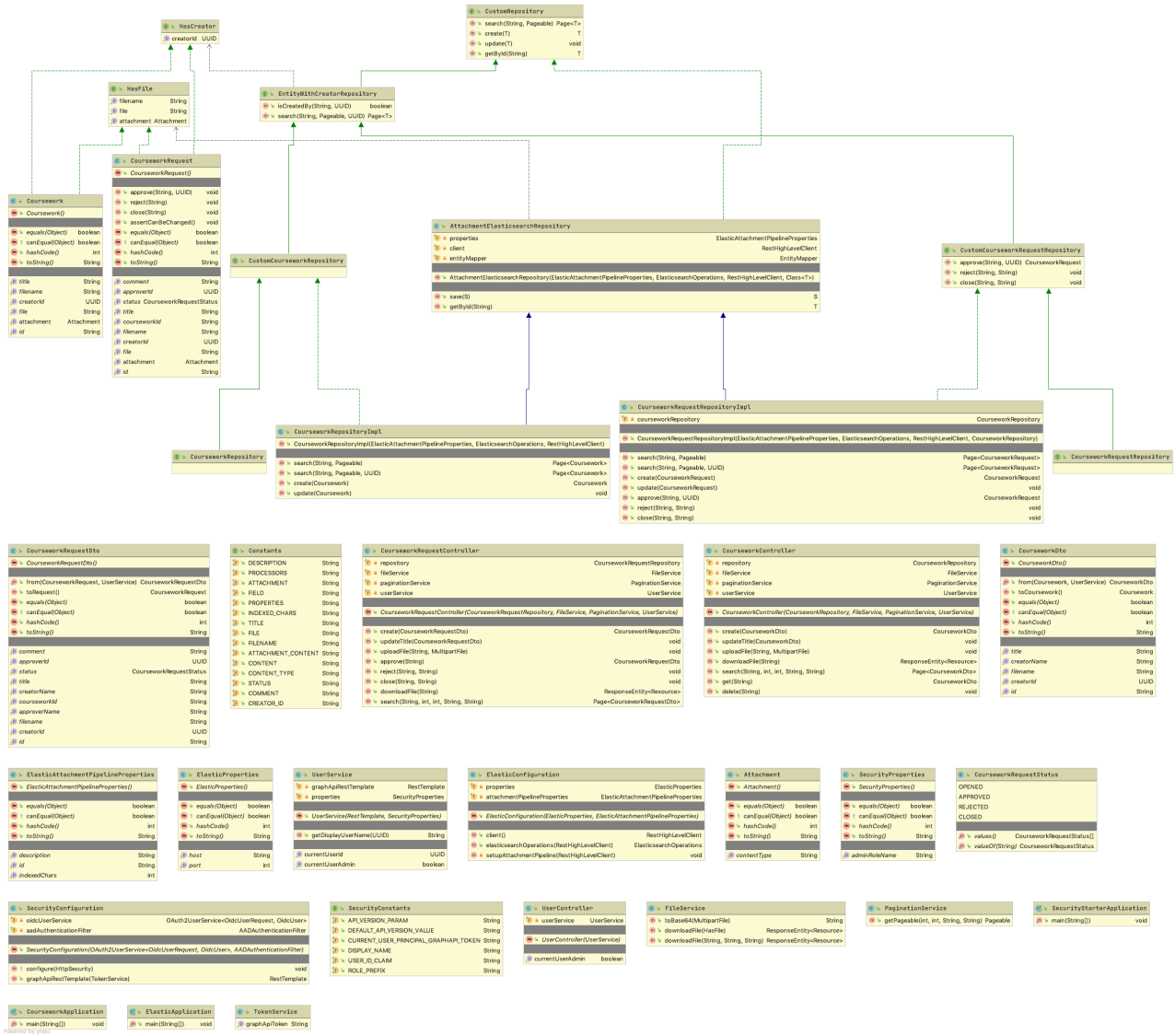

Рисунок 5.1 – Діаграма класів

5.3. Опис компонента front end

Front end компонент відповідає за відображення веб-сторінки та взаємодію користувача з сервером. При першому переході веб-сторінка перенаправляє

користувача на сторінку авторизації Azure Active Directory. Для інтеграції з Azure Active Directory було використано бібліотеку “microsoft-adal-angular6”, що підключається за допомогою npm. Для роботи цієї бібліотеки необхідно вказати певні дані вашої Azure Active Directory, які потрібні для того, щоб веб-застосунок знав, яку саме Azure Active Directory потрібно використовувати. Приклад конфігурації зображено на рисунку 5.2.

```
MsAdalAngular6Module.forRoot( adalConfig: {
  tenant: '4eb588ae-cf23-4c01-94e4-4c0f17d5effd',
  clientId: '9fa052a5-63f4-4daa-bf42-4f01ddec5f33',
  endpoints: {
    'api': 'api://c7a9cebb-acac-400a-9159-a78dc7d104f0'
  },
  redirectUri: window.location.origin,
  navigateToLoginRequestUrl: true,
  cacheLocation: 'localStorage'
}),
```

Рисунок 5.2 – Приклад конфігурації бібліотеки “microsoft-adal-angular6”

Далі потрібно вказати, для яких веб-сторінок застосунок повинен вимагати від користувача автентифікацію. У нашому випадку автентифікація необхідна для всіх сторінок, що реалізовано за допомогою http-перехоплювачів. На рисунку 5.3 можна побачити об’явлення всіх http-перехоплювачів.

```
providers: [
  { provide: HTTP_INTERCEPTORS, useClass: ApiInterceptor, multi: true },
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
],
```

Рисунок 5.3 – Об’явлення htpp-перехоплювачів

Як бачимо на рисунку 5.3, є 2 http-перехоплювача: “ApiInterceptor” та “AuthInterceptor”. Перехоплювач “AuthInterceptor” перенаправляє користувача на сторінку автентифікації, коли це потрібно; а перехоплювач “ApiInterceptor”

додає на початок шляху кожного запита на веб-сервер префікс “api”, який використовується для проксування, щоб вказати, що цей запит необхідно направити на порт веб-сервера.

5.4. Опис основного компонента

Сервер – це основний компонент, що з’єднує між собою всі інші компоненти та містить вхідні точки програми. Вхідні точки задаються за допомогою java-анотацій. Приклад вхідної точки можна побачити на рисунку 5.4.

```
@PreAuthorize("@userService.isCurrentUserAdmin()")
@PostMapping
public CourseworkDto create(@RequestBody CourseworkDto dto) {
    UUID currentUserId = userService.getCurrentUserId();

    Coursework coursework = dto.toCoursework();
    coursework.setCreatorId(currentUserId);
    return CourseworkDto.from(repository.create(coursework), userService);
}
```

Рисунок 5.4 – Вхідна точка для створення курсової роботи

Анотація “@PostMapping” вказує, що метод повинен спрацювати, коли користувач посилає REST-запит з методом “post” [5]. Шлях запита вказується за допомогою анотації “@RequestMapping”, яка знаходиться над класом, що зображено на рисунку 5.5.

```
@RestController
@RequestMapping("coursework")
@RequiredArgsConstructor
public class CourseworkController {
```

Рисунок 5.5 – Приклад використання анотації “@RequestMapping” над класом

Параметр, що передається в анотацію, вказує, що шлях запитів, які обробляється у цьому класі, повинен починатися з “coursework”. Анотація “@RestController” вказує, що цей клас містить вхідні точки, які повинні спрацьовувати на REST-запити [6]. Серверний компонент також містить модельні класи застосунку: курсова або дипломна робота, та запит на створення курсової або дипломної роботи.

Приклад модельного класа можна побачити на рисунку 5.6.

```
@Data
public class CourseworkDto {

    private String id;

    private String title;

    private String filename;

    private UUID creatorId;

    private String creatorName;
```

Рисунок 5.6 – Оголошення модельного класу “CourseworkDto”

Також серверний компонент містить сервіси для роботи з файлами та пагінацією. У майбутньому ці сервіси можуть бути винесені у окремі компоненти. Всі компоненти підключаються до основного за допомогою файлу “pom.xml”. Приклад підключення компонента для роботи з Elasticsearch можна побачити на рисунку 5.7.

```
<dependency>
    <groupId>com.igoodwill.coursework</groupId>
    <artifactId>elastic-starter</artifactId>
    <version>0.0.1</version>
</dependency>
```

Рисунок 5.7 – Підключення компонента для роботи з Elasticsearch

Конфігурація, необхідна для різних компонентів, знаходиться в файлі “application.properties”, який зображено на рисунку 5.8.

```
# Elastic starter
com.igoodwill.coursework.elastic.host=localhost
com.igoodwill.coursework.elastic.port=9200
# Security starter
com.igoodwill.coursework.security.admin-role-name=Admins
```

Рисунок 5.8 – Приклад конфігурації компонентів у файлі “application.properties”

5.5. Опис компонента для роботи з автентифікацією і авторизацією

Компонент для роботи з автентифікацією і авторизацією містить сервіси для отримання інформації про користувача. Для роботи з Azure Active Directory використовується “azure-active-directory-spring-boot-starter”. На рисунку 5.9 зображено, як відбувається отримання імені користувача за його ідентифікатором.

```
public String getDisplayUserName(UUID userId) {
    return Optional
            .ofNullable(userId) Optional<UUID>
            .map(id -> "/users/" + id) Optional<String>
            .map(url -> graphApiRestTemplate.getForEntity(url, Map.class)) Optional<ResponseEntity<Map>>
            .map(HttpEntity::getBody) Optional<Map>
            .map(body -> (String) body.get(DISPLAY_NAME)) Optional<String>
            .orElse( other: null);
}
```

Рисунок 5.9 – Отримання імені користувача за ідентифікатором

На рисунку 5.10 зображено, як відбувається перевірка, чи має користувач права адміністатора.

```
public boolean isCurrentUserAdmin() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();
    Set<String> authoritiesSet = AuthorityUtils.authorityListToSet(authorities);
    return authoritiesSet.contains(ROLE_PREFIX + properties.getAdminRoleName());
}
```

Рисунок 5.10 – Перевірка прав адміністатора у користувача

5.6. Опис компонента для роботи з Elasticsearch

Компонент для роботи з Elasticsearch містить весь код, необхідний для читання та запису в Elasticsearch. Під час запуску програми створюється “pipeline” для обробки текстових файлів різного формату таким чином, щоб користувач міг здійнювати пошук за їх вмістом. На стороні Elasticsearch-сервера для цього використовується плагін “attachment-pipeline”. Сторення “pipepine” зображено на рисунку 5.11.

```
JSONArray properties = new JSONArray();
properties.add(CONTENT);
properties.add(CONTENT_TYPE);

JSONObject attachment = new JSONObject();
attachment.put(FIELD, FILE);
attachment.put(INDEXED_CHARS, attachmentPipelineProperties.getIndexedChars());
attachment.put(PROPERTIES, properties);

JSONObject processor = new JSONObject();
processor.put(ATTACHMENT, attachment);

JSONArray processors = new JSONArray();
processors.add(processor);

JSONObject source = new JSONObject();
source.put(DESCRIPTION, attachmentPipelineProperties.getDescription());
source.put(PROCESSORS, processors);

client
        .ingest()
        .putPipeline(
                new PutPipelineRequest(
                        attachmentPipelineProperties.getId(),
                        new BytesArray(source.toString().getBytes(StandardCharsets.UTF_8))
                        XContentType.JSON
                ),
                RequestOptions.DEFAULT
        );
```

Рисунок 5.11 – Створення “pipeline” для обробки текстових файлів

При збереженні файлів курсових та дипломних робіт також створюються індекси, що можна побачити на рисунку 5.12.

```
@Override
public <S extends T> S save(@NonNull S entity) {
    String indexName = entityInformation.getIndexName();
    String documentId;
    try {
        IndexRequest indexRequest = new IndexRequest()
                .index(indexName)
                .type(entityInformation.getType())
                .id(entityInformation.getId(entity))
                .source(entityMapper.mapToString(entity), Requests.INDEX_CONTENT_TYPE);

        if (entity.getFile() != null) {
            indexRequest.setPipeline(properties.getId());
        }

        documentId = client.index(indexRequest, RequestOptions.DEFAULT).getId();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    ElasticsearchPersistentEntity<?> persistentEntity = elasticsearchOperations.getPersistentEntityFor(entity.getClass());
    ElasticsearchPersistentProperty idProperty = persistentEntity.getIdProperty();
    if (idProperty != null && idProperty.getType().isAssignableFrom(String.class)) {
        persistentEntity.getPropertyAccessor(entity).setProperty(idProperty, documentId);
    }

    elasticsearchOperations.refresh(indexName);
    return entity;
}
```

Рисунок 5.12 – Створення індексів файлів курсових та дипломних робіт

На рисунку 5.13 зображено пошук за назвою курсової роботи, за назвою та вмістом файла курсової роботи.

```
@Override
public Page<Coursework> search(String searchQuery, Pageable pageable, UUID userId) {
    BoolQueryBuilder query = QueryBuilders
            .boolQuery()
            .must(
                    QueryBuilders
                            .queryStringQuery("*" + QueryParser.escape(searchQuery) + "*")
                            .field(TITLE)
                            .field(FILENAME)
                            .field(ATTACHMENT_CONTENT)
            );

    if (userId != null) {
        query.must(QueryBuilders.matchQuery(CREATOR_ID, userId.toString()));
    }

    return search(query, pageable);
}
```

Рисунок 5.13 – Пошук курсових робіт за запитом

5.7. Налаштування системи

Налаштування відбувається в файлі “app/src/main/resources/application.properties”:

а) “com.igoodwill.coursework.elastic.host” – адреса Elasticsearch сервера.

б) “com.igoodwill.coursework.elastic.port” – порт Elasticsearch сервера.

в) “com.igoodwill.coursework.security.admin-role-name” – ім’я ролі адміністратора.

г) “azure.activedirectory.*” – відповідні параметри Azure Active Directory.

5.8. Розгортка системи

Для розгортки необхідно:

а) Встановити Elasticsearch та плагін “attachment-pipeline”.

б) Запустити Elasticsearch, оновити адресу та порт в файлі “application.properties”.

в) Створити Azure Active Directory, налуштувати ролі користувачів та оновити відповідні параметри в файлі “application.properties”.

г) Запустити CourseworkApplication.java.

е) Перейти в директорію “front” та виконати команду “npm start”.

РОЗДІЛ 6: Використання системи

Після розгортки системи веб-сторінка стане доступною за адресою “localhost:4200”. При першому заході на веб-сторінку, користувача буде перенаправлено на сторінку автентифікації, яку можна побачити на рисунку 6.1.

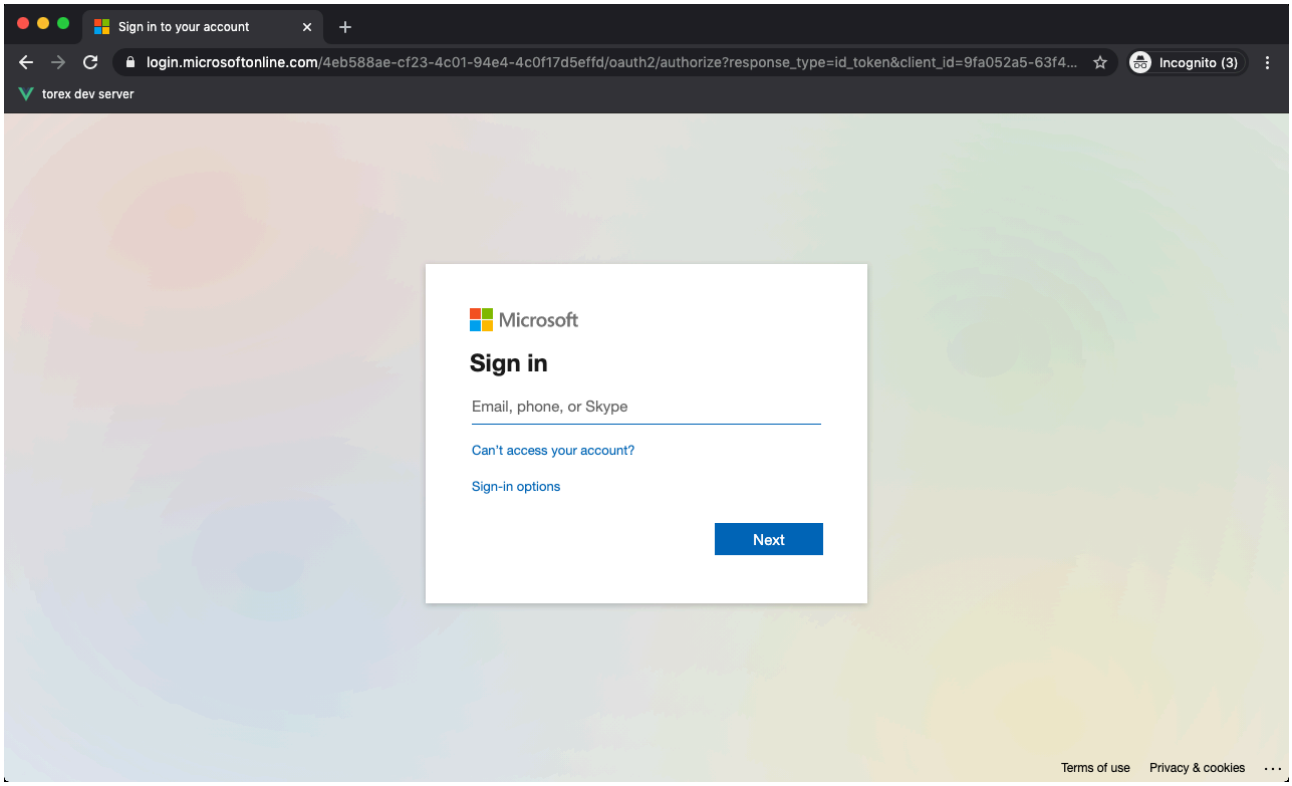

Рисунок 6.1 – Сторінка автентифікації

Після проходження автентифікації відкриється сторінка дипломних та курсових робіт, яку можна побачити на рисунку 6.2.

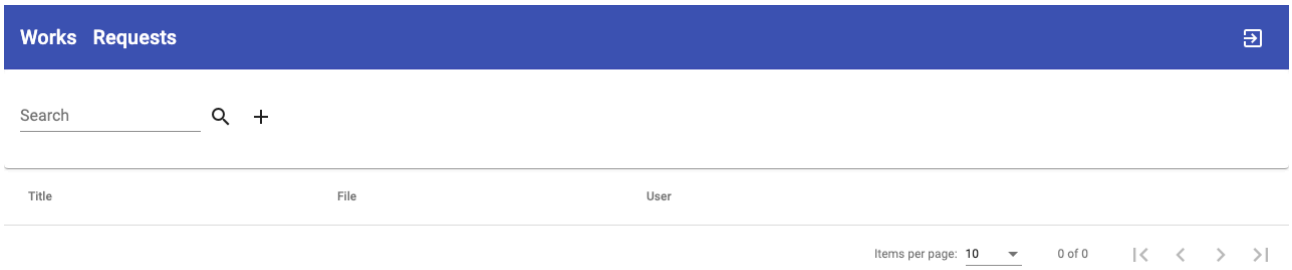

Рисунок 6.2 – Сторінка дипломних та курсових робіт

В верхній правій частині сторінки знаходиться кнопка для зміни користувача. В верхній лівій частині сторінки знаходиться навігація для

переходу між сторінками робіт та запитів на створення робіт. В основній частині сторінки є поле для пошуку, кнопка для створення нової роботи та таблиця всіх робіт. Після натискання кнопки для створення нової роботи відкривається діалогове вікно, яке зображено на рисунку 6.3.

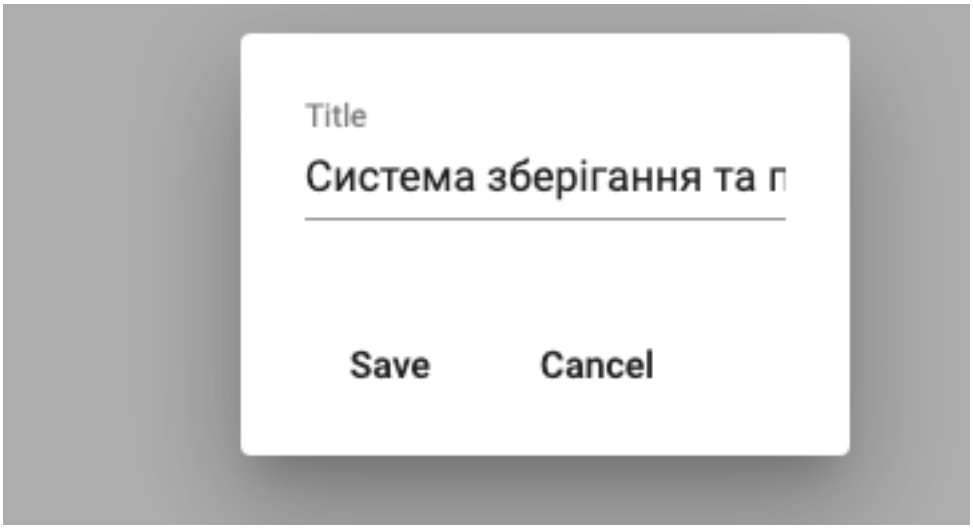

Рисунок 6.3 – Діалогове вікно для створення нової роботи

У діалоговому вікні потрібно ввести назву роботу та натиснути кнопку “Save”, після чого робота з’явиться у таблиці. Оновлену таблицю можна побачити на рисунку 6.4.

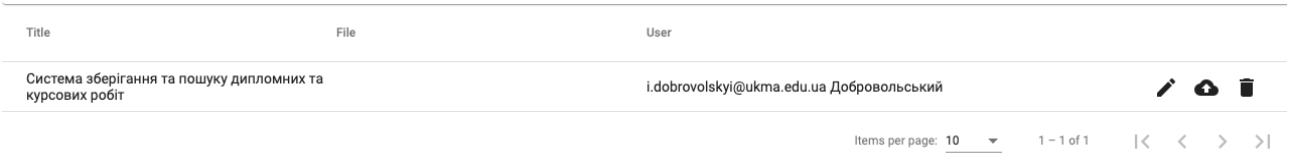

Рисунок 6.4 – Таблиця з одною роботою

В таблиці робіт можна побачити назву роботи, файл та автора, а також кнопки для редагування назви, завантаження файлу та видалення роботи. Діалогове вікно для завантаження файлу роботи зображено на рисунку 6.5.

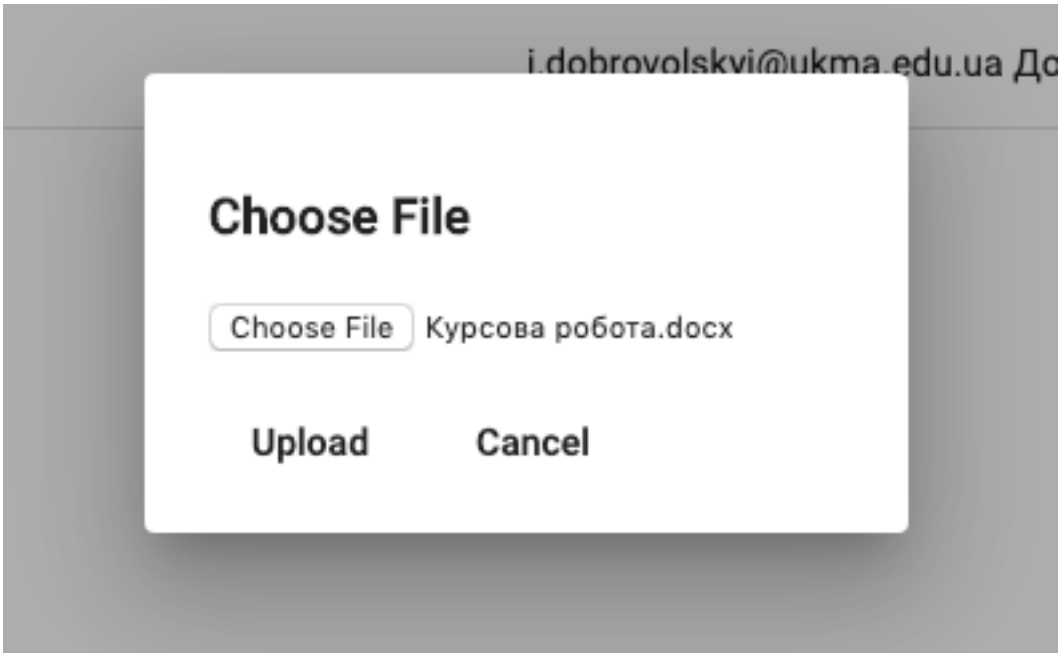

Рисунок 6.5 – Діалогове вікно для завантаження файлу роботи

Після завантаження файлу роботи в таблиці з’явиться посилання на цей файл. Таблиця з завантаженим файлом зображена на рисунку 6.6.

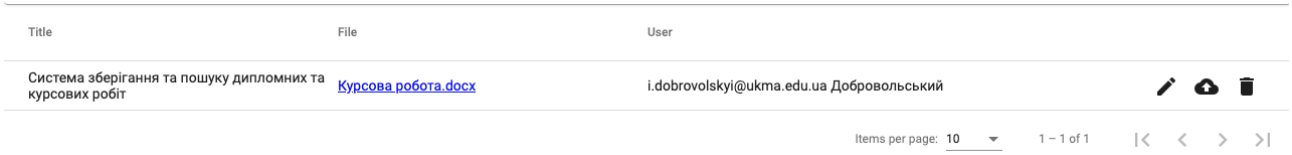

Рисунок 6.6 – Таблиця з завантаженим файлом

Кожну роботу можна також переглянути на її персональній сторінці. Для переходу на сторінку роботи потрібно натиснути на назву роботи в таблиці. Цю сторінку можна побачити на рисунку 6.7.

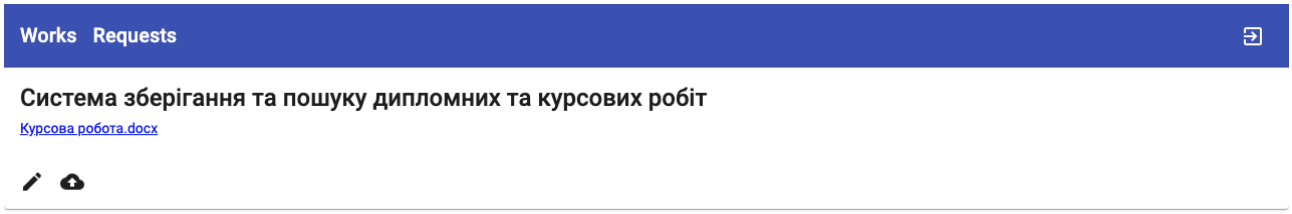

Рисунок 6.7 – Сторінка роботи

Сторінка запитів на створення робіт має вигляд аналогічний до сторінки робіт. У діалоговому вікні для створення потрібно ввести назву роботи. Після створення можна завантажити файл роботи. Сторінка запитів на створення робіт зображена на рисунку 6.8.

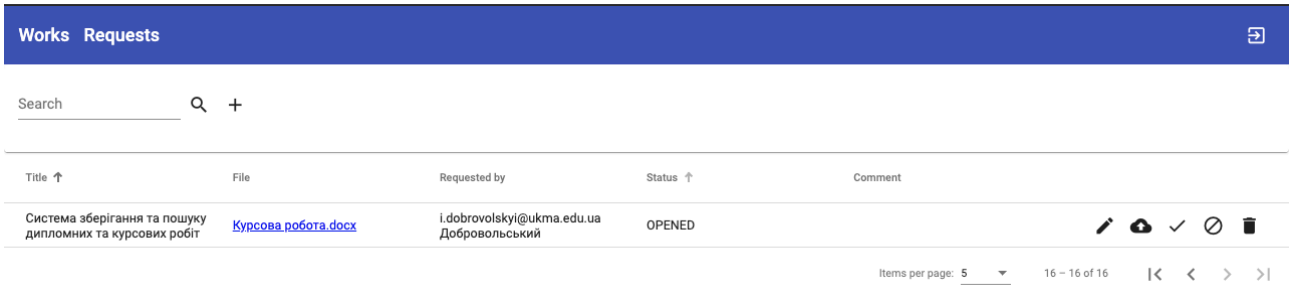

Рисунок 6.8 – Сторінка запитів на створення робіт

В таблиці запитів також можна також побачити статус за коментар до запиту. Можливі статуси запиту: запит створено, запит підтверджено, запит скасовано, запит закрито. Після підтвердження запиту буде створено відповідну роботу. Для скасування запиту необхідно ввести причину. Діалогове вікно для скасування запиту зображено на рисунку 6.9.

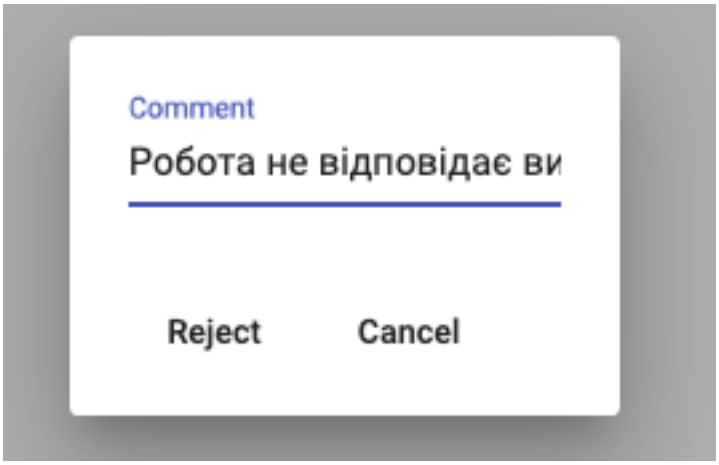

Рисунок 6.9 – Діалогове вікно для скасування запиту

Висновки

В процесі виконанні курсової роботи було створено прототип веб-застосунку для зберігання та пошуку дипломних та курсових робіт. Для швидкого пошуку за вмістом робіт було вирішено індексувати всі файли в момент збереження, для чого було використано сервер Elasticsearch, який надає можливість зберігати, індексувати документи та здійснювати пошук за запитом. Для того, щоб була можливість працювати с файлами різних форматів (таких як pdf або doc), на сервер Elasticsearch також було встановлено плагін “attachment-pipeline”. Для роботи цього плагіна необхідно було створити так званий “pipeline”, який вказує, які саме поля документу потрібно індексувати з використанням плагіна. Також було реалізовано автентифікацію та авторизацію. Для того, щоб була можливість використовувати готову базу користувачів (наприклад, студентів університету), було реалізовано інтеграцію з сервісом Azure Active Directory. Серверну частину було реалізовано на базі Spring Framework, а веб-сторінку - з використанням фреймворку Angular.

Список використаної літератури

1. https://whatis.techtarget.com/definition/multi-tenancy
2. https://lucene.apache.org/solr/features.html
3. https://spring.io/web-applications
4. https://angular.io/docs
5. https://docs.spring.io/spring/docs/4.3.26.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf (сторінки 491-492)
6. https://docs.spring.io/spring/docs/4.3.26.RELEASE/spring-framework-reference/pdf/spring-framework-reference.pdf (сторінка 506)