Document Management and Q&A Application

📁 **Project Structure**

com.example.docqa

├── config // Swagger, Security config

├── controller // REST API controllers

├── exception // Custom exception handling

├── model // Entity classes

├── repository // Spring Data JPA interfaces

├── service // Business logic interfaces & implementations

├── util // Utility classes

├── DocQaApplication.java // Main Spring Boot application class

---

✅ **Key Functionalities**

1. Document Upload API

- Endpoint: POST /api/documents/upload
- Accepts: Multipart file, author, type
- Parses file (txt, pdf, docx) using Apache Tika and stores content in DB.

2. Search API (Q&A)

- Endpoint: GET /api/documents/search?query=...
- Accepts a query keyword and returns relevant document snippets.

3. Filter + Pagination API

- Endpoint: GET /api/documents/filter
- Supports filtering by author, type, with pagination and sorting.

4. Swagger UI

- View API documentation at: http://localhost:8080/swagger-ui.html

Test Case Documentation for DocumentServiceImpl

🔹 **Test Class: DocumentServiceImplTest**

Package: com.example.docqa.service
Purpose: To validate core document service functionalities including upload, search, async save, and filtering logic.

✅ **1. testSaveDocument_successfulUpload**

- **Purpose:** Ensure a document is correctly parsed, saved, and returned with a success message.

- **Input:** MultipartFile (PDF content), author: "Pradip", type: "PDF"
- **Expected Output:**
    - Message: "Document uploaded successfully"
    - Document saved with correct author, type, and snippet.
- **Assertions:**
    - Assert message.
    - Assert document repository save call.
    - Assert author, type, content, and upload date are set.

❌ 2. testSaveDocument_throwsIOException

- **Purpose:** Validate exception handling during file upload when an IOException occurs.
- **Input:** MultipartFile that throws IOException on getInputStream()
- **Expected Output:** RuntimeException with message "Error reading document content"
- **Assertions:**
    - Asserts RuntimeException is thrown.
    - Asserts exception message contains expected text.

🔍 3. testSearchDocuments_returnsResults

- **Purpose:** Ensure keyword-based search returns valid document snippets.
- **Input:** Keyword "test" with matching document in mock repository.
- **Expected Output:**
    - One document in result.
    - Snippet includes keyword.
- **Assertions:**
    - Check result size.
    - Validate snippet, author, and type values.

🔍 4. testSearchDocuments_emptyResults

- **Purpose:** Verify search returns empty list when no matches found.
- **Input:** Keyword "no-match"
- **Expected Output:** Empty list.
- **Assertions:** Assert list size is zero.

🔄 5. testSaveDocumentAsync

- **Purpose:** Test asynchronous version of document upload.
- **Input:** Async MultipartFile "hello.txt" with content.
- **Expected Output:** CompletableFuture completed with success message.
- **Assertions:**
  - future.isDone() is true.
  - Message is "Document uploaded successfully".

📑 6. testFilterDocuments

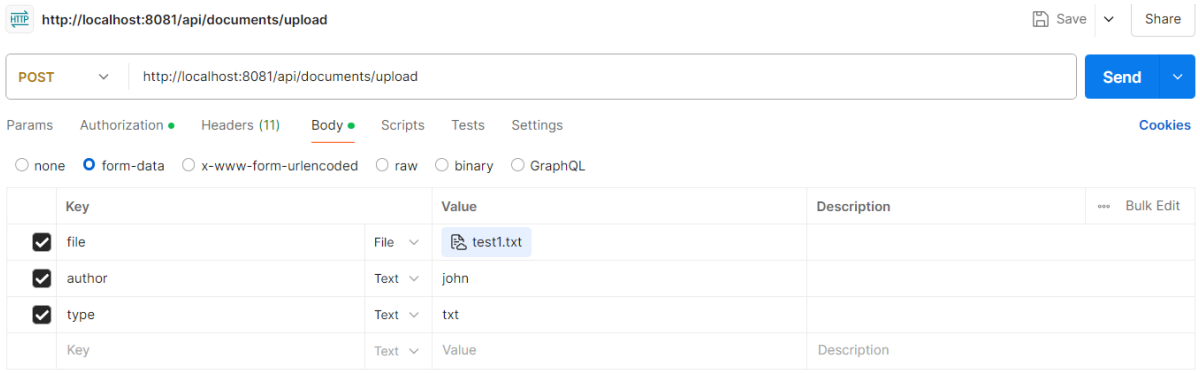
- **Purpose:** Test document filtering with pagination and sorting.
- **Input:**
  - Author: "Sam", Type: "pdf", Page: 0, Size: 10, SortBy: "uploadDate", SortDir: "desc"
- **Expected Output:**
  - One document with correct metadata.
  - Page number 0, Total pages 1.
- **Assertions:**
  - Size of document list is 1.
  - Check current page and total page.

🍼 Summary Table

Test Method	Scenario	Status
testSaveDocument_successfulUpload	Valid upload saves document	✅ Pass
testSaveDocument_throwsIOException	File read failure	✅ Pass
testSearchDocuments_returnsResults	Keyword search returns result	✅ Pass
testSearchDocuments_emptyResults	No search result	✅ Pass
testSaveDocumentAsync	Async document save	✅ Pass
testFilterDocuments	Filter by metadata, pagination, sorting	✅ Pass

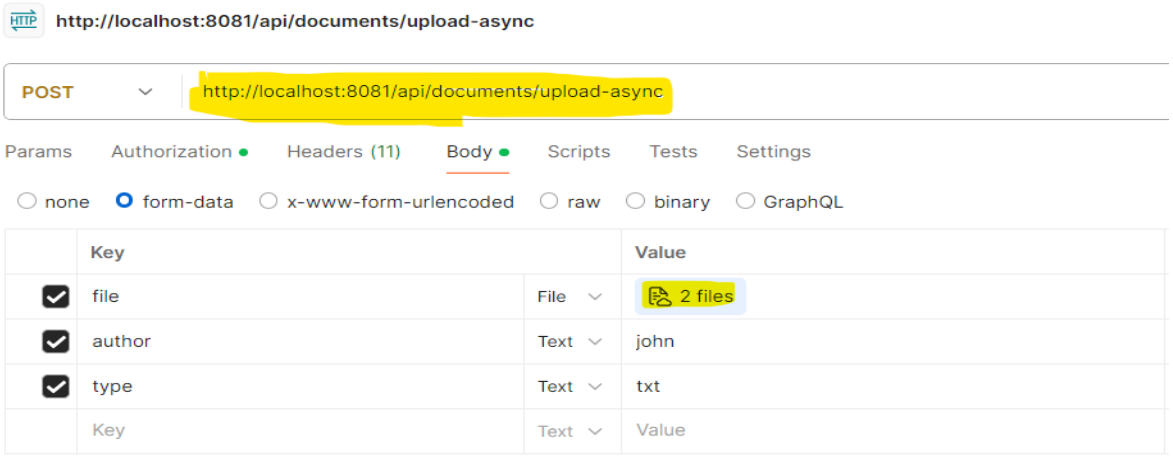
First Test

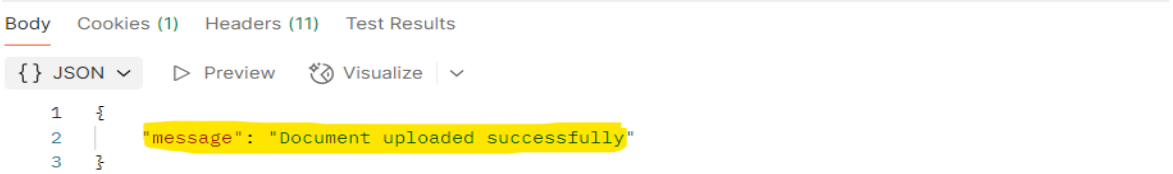
Post url for upload one file add time



Second test

For upload multiple same time like async proceed





Third Test Case case

Search base any message like sample