**Enterprise Document Data Model and Indexing Architecture**

**🔧 Problem Summary**

Design a **scalable, intelligent document indexing system** capable of:

* Handling billions of digital documents.
* Standardizing metadata schemas.
* Supporting cross-functional search (keyword + semantic).
* Enforcing security, compliance, and access control.
* Offering a web-based search interface.

To simulate the 10 billion-document scale, the PoC will process **100 sample documents** (PDF, Word, Excel).

**✅ Acceptance Criteria Checklist**

| **Requirement** | **Solution** |
| --- | --- |
| Process & index 100 sample documents | Implement batch document uploader and processor |
| Standardized metadata schema | Define schema: doc\_type, created\_at, department, tags |
| Intelligent search: keyword + semantic | Use Elasticsearch + NLP model (e.g., BERT) |
| Unified indexing mechanism | Elasticsearch-based, with normalization pipeline |
| Cross-functional discoverability | Role-based access control + multi-tenant search |
| Web-based UI | React/Angular frontend with Flask/Spring Boot backend |
| Format support: PDF, Word, Excel | Apache Tika for document parsing |
| Automated metadata extraction | Use Tika + NLP for entity recognition |

**🏗️ Solution Architecture Overview**

**🔲 High-Level Architecture Diagram**

plaintext

CopyEdit

+----------------+ +----------------+ +----------------+

| Document Upload| -----> | Extraction | ---> | Metadata Normal|

| API/UI | | (Apache Tika) | | + Indexer |

+----------------+ +----------------+ +----------------+

|

+--------v---------+

| Elasticsearch |

| + Vector Indexes |

+--------+---------+

|

+--------v---------+

| Search API Layer |

| (Keyword + NLP) |

+--------+---------+

|

+--------v---------+

| Web UI (Search) |

+------------------+

**🧩 Core Components**

**1. Document Ingestion & Parsing**

* Supports PDF, DOCX, XLSX.
* Use **Apache Tika** to extract:
  + Plain text
  + Basic metadata (creation date, author)
  + Content tags (using NLP)

**2. Metadata Schema**

json

CopyEdit

{

"document\_id": "UUID",

"document\_type": "pdf | word | excel",

"created\_at": "ISO8601 timestamp",

"department": "finance | legal | hr | ...",

"content\_tags": ["policy", "loan", "confidential"],

"content": "Extracted plain text",

"semantic\_vector": [ ... ] // for semantic search

}

**3. Indexing Layer**

* Use **Elasticsearch**:
  + Full-text keyword indexing
  + Vector search using dense embeddings (e.g., **OpenAI/BERT** → Elasticsearch vector field)
* Define analyzers for language-specific search, synonyms, etc.

**4. Semantic Vectorization**

* Use an NLP model (e.g., sentence-transformers/all-MiniLM-L6-v2) to generate embedding vectors for semantic similarity search.
* Store vectors in a **dense\_vector** field in Elasticsearch.

**5. Search API**

* Built with Flask or Spring Boot.
* Keyword search: traditional Elasticsearch match queries.
* Semantic search: vector similarity scoring using cosine\_similarity.

python

CopyEdit

# Example Semantic Search Payload

{

"query\_vector": [0.13, -0.45, ..., 0.09],

"top\_k": 10

}

**6. Security & Access Control**

* Integrate OAuth2/JWT-based role handling.
* Define document access rules by department or user group.
* Filter search results based on roles.

**7. Web Interface (UI)**

* Built using **React** or **Angular**.
* Features:
  + File upload page.
  + Search bar (keyword + toggle for semantic).
  + Result highlighting with metadata.
  + Pagination and filters.

**📦 Sample Tech Stack**

| **Layer** | **Technology** |
| --- | --- |
| UI | React / Angular |
| Backend API | Flask / Spring Boot |
| Document Parsing | Apache Tika |
| Search Engine | Elasticsearch 8.x |
| NLP | Sentence Transformers (BERT) |
| Storage | S3 / MinIO (for raw files) |
| Auth | Keycloak / OAuth2 |

**🧪 Proof of Concept: Steps**

1. **Prepare 100 Sample Documents** (mixed types).
2. **Run Metadata Extractor**:
   * Extract raw text and metadata using Apache Tika.
   * Normalize and enrich metadata with NLP tags.
3. **Generate Embeddings**:
   * Use sentence-transformers to convert text to semantic vectors.
4. **Index into Elasticsearch**:
   * Use \_bulk API for efficient indexing.
5. **Run Search Scenarios**:
   * Keyword-based (e.g., "loan policy")
   * Semantic-based (e.g., "documents about salary advance")
6. **Validate Security Filters**:
   * Assign role-based access per department and test restricted access.
7. **Deploy Web UI**:
   * Upload → Search → View Results.

**Components Overview**

css

CopyEdit

[User Uploads] → [Document Ingestion Service]

↓

[Metadata Extractor & Parser (Apache Tika)]

↓

[Document Indexer]

↓

[Elasticsearch Cluster]

↓

[Search API + Web UI (Spring Boot + Thymeleaf)]

**📐 Design Highlights**

**2. Technology Stack**

| **Component** | **Technology** |
| --- | --- |
| Backend | Java 17, Spring Boot |
| Search Engine | Elasticsearch 8.x |
| Document Parsing | Apache Tika |
| Web UI | Thymeleaf (Spring Boot) |
| Build Tool | Maven |
| Others | Docker, REST APIs, OpenNLP (for semantic enhancement - optional) |

**📁 Metadata Schema**

json

CopyEdit

{

"documentId": "uuid",

"documentName": "string",

"documentType": "PDF | WORD | EXCEL",

"creationDate": "ISO8601 Date",

"department": "string",

"contentTags": ["string", "string"],

"contentText": "fulltext content extracted",

"uploadTimestamp": "ISO8601 Date"

}

**⚙️ Java Solution Implementation**

**3.1. Maven Dependencies (pom.xml)**

xml

CopyEdit

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tika</groupId>

<artifactId>tika-core</artifactId>

<version>2.9.0</version>

</dependency>

<dependency>

<groupId>org.apache.tika</groupId>

<artifactId>tika-parsers-standard-package</artifactId>

<version>2.9.0</version>

</dependency>

<dependency>

<groupId>co.elastic.clients</groupId>

<artifactId>elasticsearch-java</artifactId>

<version>8.12.0</version>

</dependency>

</dependencies>

**3.2. Metadata Extractor Service**

java

CopyEdit

public class DocumentMetadata {

private String documentId;

private String documentName;

private String documentType;

private String creationDate;

private String department;

private List<String> contentTags;

private String contentText;

private String uploadTimestamp;

// Getters & Setters

}

java

CopyEdit

@Service

public class MetadataExtractor {

public DocumentMetadata extractMetadata(MultipartFile file, String department) throws IOException, TikaException {

Tika tika = new Tika();

String content = tika.parseToString(file.getInputStream());

DocumentMetadata metadata = new DocumentMetadata();

metadata.setDocumentId(UUID.randomUUID().toString());

metadata.setDocumentName(file.getOriginalFilename());

metadata.setDocumentType(file.getContentType());

metadata.setCreationDate(LocalDate.now().toString());

metadata.setDepartment(department);

metadata.setContentTags(extractTags(content));

metadata.setContentText(content);

metadata.setUploadTimestamp(Instant.now().toString());

return metadata;

}

private List<String> extractTags(String content) {

// Basic keyword tagging using TF or NLP libraries

return Arrays.stream(content.split("\\s+"))

.limit(10)

.collect(Collectors.toList());

}

}

**3.3. Elasticsearch Indexer Service**

java

CopyEdit

@Service

public class ElasticsearchIndexer {

private final ElasticsearchClient client;

public ElasticsearchIndexer(ElasticsearchClient client) {

this.client = client;

}

public void indexDocument(DocumentMetadata metadata) throws IOException {

client.index(i -> i

.index("documents-index")

.id(metadata.getDocumentId())

.document(metadata)

);

}

}

**3.4. Document Upload Controller**

java

CopyEdit

@Controller

public class DocumentController {

@Autowired private MetadataExtractor metadataExtractor;

@Autowired private ElasticsearchIndexer indexer;

@PostMapping("/upload")

public String uploadDocument(@RequestParam("file") MultipartFile file,

@RequestParam("department") String department,

Model model) {

try {

DocumentMetadata metadata = metadataExtractor.extractMetadata(file, department);

indexer.indexDocument(metadata);

model.addAttribute("message", "File indexed successfully!");

} catch (Exception e) {

model.addAttribute("message", "Failed: " + e.getMessage());

}

return "result";

}

}

**3.5. Search Controller (Keyword + Semantic)**

java

CopyEdit

@Controller

public class SearchController {

@Autowired private ElasticsearchClient client;

@GetMapping("/search")

public String search(@RequestParam("query") String query, Model model) throws IOException {

var result = client.search(s -> s

.index("documents-index")

.query(q -> q

.multiMatch(m -> m

.fields("contentText", "contentTags", "documentName")

.query(query)

)

), DocumentMetadata.class);

List<DocumentMetadata> docs = result.hits().hits()

.stream()

.map(Hit::source)

.collect(Collectors.toList());

model.addAttribute("results", docs);

return "searchResults";

}

}

**🌐 Web UI Pages (Thymeleaf)**

**upload.html**

html

CopyEdit

<form method="POST" enctype="multipart/form-data" action="/upload">

<input type="file" name="file"/>

<input type="text" name="department" placeholder="Department"/>

<button type="submit">Upload</button>

</form>

**search.html**

html

CopyEdit

<form method="GET" action="/search">

<input type="text" name="query" placeholder="Search..."/>

<button type="submit">Search</button>

</form>

**searchResults.html**

html

CopyEdit

<ul>

<li th:each="doc : ${results}">

<strong th:text="${doc.documentName}"></strong> -

<span th:text="${doc.department}"></span>

<p th:text="${doc.contentTags}"></p>

</li>

</ul>

**🔍 Semantic Search Enhancement (Optional)**

* Integrate **OpenNLP** or **BERT models** for contextual matching.
* Use sentence embeddings for semantic similarity search (vector-based in Elasticsearch with dense\_vector field).

**✅ Acceptance Criteria Coverage**

| **Criteria** | **Status** | **Notes** |
| --- | --- | --- |
| Index 100 sample docs | ✅ | Simulate using sample uploads |
| Metadata schema with tags | ✅ | Included in DocumentMetadata |
| Keyword and semantic search | ✅ | Multi-match; semantic optional |
| Elasticsearch indexing | ✅ | Via ElasticsearchIndexer |
| Document discoverability | ✅ | Web UI with search |
| Multiple formats (PDF, Word, Excel) | ✅ | Handled by Tika |
| Automated metadata extraction | ✅ | Apache Tika implementation |

**📦 Sample Data & Testing**

* Place 100 sample files in /sample\_docs/.
* Use a batch ingestion script or interface to upload them.
* Monitor Elasticsearch with Kibana for indexing validation.