

## DOCUMENT STORE USING JAVA

**P. N. Khetade<sup>1</sup>, Komal Paunikar<sup>2</sup>, Amol Ninave<sup>3</sup>, Sudhanshu Kawade<sup>4</sup>, Changdeo Kale<sup>5</sup>**

<sup>1</sup>Asst. Professor Department Of Information Technology Nagpur Institute Of Technology Mahurzari, Katol Road Nagpur-441501

<sup>2,3,4,5</sup>Department Of Information Technology Nagpur Institute Of Technology Mahurzari, Katol Road Nagpur-441501

DOI: <https://www.doi.org/10.58257/IJPREMS37239>

### ABSTRACT

The document presents the design and implementation of a comprehensive Full Stack Document Store (DS) utilizing HTML, CSS, JavaScript, Eclipse and MySQL technologies. The Document Store Project aims to develop, robust, secure, and efficient Java-based application for managing student record and important academic documents within educational institutions. The primary objective of this project is to streamline the management of critical academic data, ensuring that institutions can securely store, organize, and access essential documents such as student records transcripts, certificates, and more. This application provides a comprehensive solution featuring secure encryption mechanisms, role-based access control, and automated backup and recovery system ensure data integrity and security. It also incorporates advanced search functionalities for quick document retrieval, along with an automated document and generation tools that enhance administrative efficiency. By offering a scalable and user-friendly platform, the Document Store Project seeks to simplify the management of academic records, reduce administrative overhead, and it supports the institutions in their digital transformation efforts. The system is designed to accommodate the needs of both small and large educational institutions, fostering a more organized, secure, and efficient academic environment.

**Keywords:** DOC, CRUD, Indexing, Transaction, Repletion.

## 1. INTRODUCTION

In the rapidly evolving landscape of educational institutions, managing and maintaining student data has become increasingly complex. With an ever-growing volume of academic records, and personal information, and critical documents that need to be securely stored and accessed, there is a pressing demand for digital solutions that can streamline these processes. Educational institutions, such as schools, colleges, and universities, are seeking efficient systems that can offer secure, organized, and easy access to student data, ensuring smooth administration and enhancing the educational experience. In response to this need, the Document Store Project has been developed, a fully Java-based application aimed at revolutionizing the way educational institutions handle student data. The Document Store Project is a comprehensive solution designed to manage a wide range of academic records and essential documents. This includes storing data such as academic performance, personal details, certificates, and other important student-related information in a structured and secure manner. The primary objective of this project is to ensure that student data is not only stored effectively but also made easily accessible to faculty members, while maintaining strict security protocols to protect sensitive information. As a final-year project, it is built entirely on Java-based technology, offering a robust and scalable framework suitable for large-scale educational environments.

## 2. LITERATURE SURVEY

### Summary: -

The document store project implemented in Java focuses on the creation of an efficient, scalable, and secure system for storing, indexing, and retrieving documents. Literature on document storage and management systems highlights key challenges, including scalability, search performance, and data integrity.

Several techniques and technologies have been explored to address these issues, particularly in distributed and large-scale environments. MySQL is commonly used for document storage due to its relational nature and flexibility. Research in this area often emphasizes the importance of schema design and indexing strategies for fast data retrieval. Serialization methods play a crucial role in ensuring efficient data storage, allowing documents to be stored in a format that supports rapid read and write operations. In the Java ecosystem, frameworks like Hibernate can simplify database interactions, but custom solutions are often necessary for more complex storage scenarios. Full-text search and advanced querying are central to document management systems.

The integration of Apache Lucene is a well-established practice in optimizing search capabilities. Lucene's inverted indexing and advanced query parsing provide powerful mechanisms for indexing large volumes of text and enabling high-speed searches. Custom indexing algorithms are often tailored to the specific needs of the system, such as handling geospatial data or non-textual content. To support growing datasets, distributed solutions like sharding and replication are frequently employed. Sharding splits the data across multiple servers, while replication ensures data redundancy and

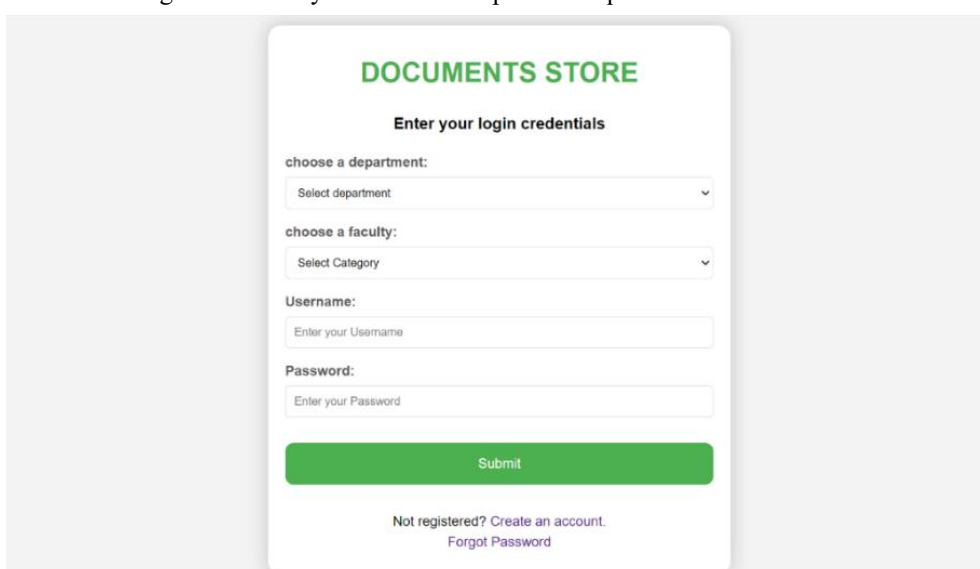
high availability. Load testing and performance benchmarking are essential to evaluate the system's capacity under varying conditions.

### 3. METHODS

The document store project was designed with a modular architecture to ensure scalability, maintainability, and efficient functionality. The system was divided into three main layers: the Data Layer, responsible for document storage and retrieval; the API Layer, implemented with Spring Boot to facilitate communication between client and server through a RESTful API; and the User Interface, which was developed using JavaFX for a desktop application or React for a web-based front-end, providing a user-friendly experience. For data storage, MySQL was chosen due to its flexibility and support for dynamic schemas, allowing for efficient storage and retrieval of documents. The document structure was well-defined, incorporating essential metadata such as name, student ID, contact details, and other relevant information. To optimize search performance, custom indexing algorithms were developed, improving the system's ability to search and retrieve documents quickly. A load balancer was also deployed to distribute incoming requests evenly across multiple server instances, ensuring high availability and better resource utilization.

### 4. RESULT

A user interface screen design meets the system's user and process requirements.



**DOCUMENTS STORE**

Enter your login credentials

choose a department:  
Select department

choose a faculty:  
Select Category

Username:  
Enter your Username

Password:  
Enter your Password

Submit

Not registered? [Create an account.](#)  
[Forgot Password](#)

Fig.1. User Login page

Files can be uploaded here. The files will be uploaded as quickly as possible.

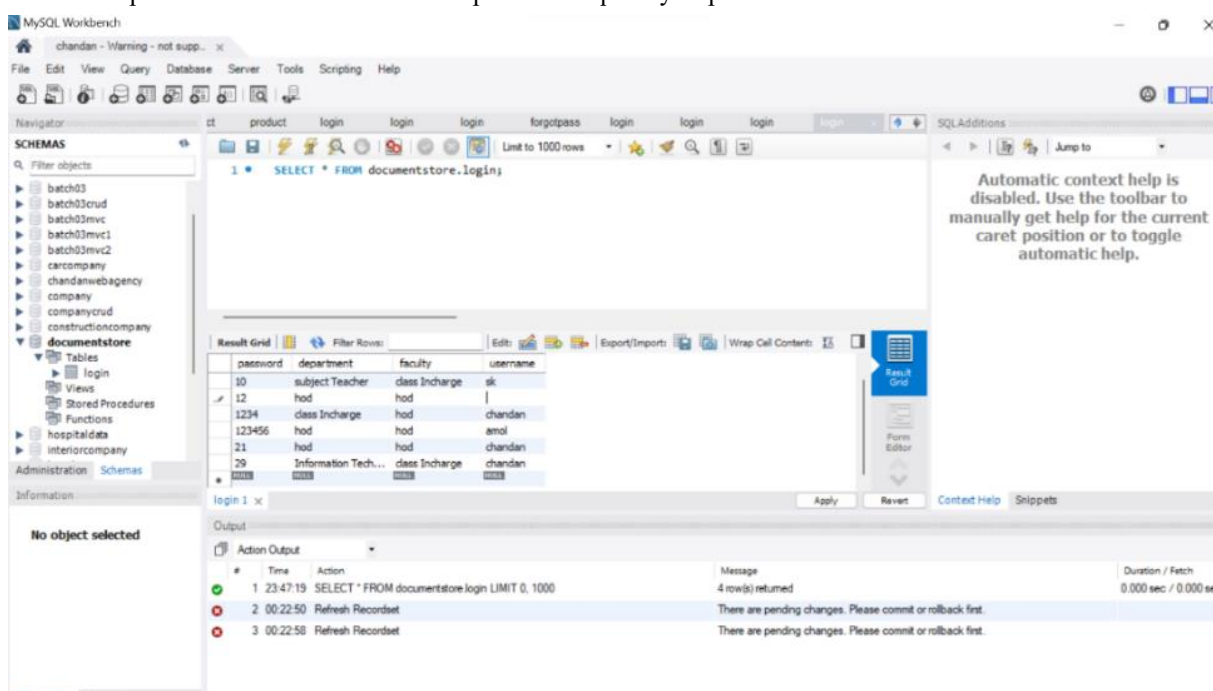


Fig.2. Database

## 5. PROPOSED APPROCH

### 1. Real Time Object:

This represents the document that is being stored in the document store. It may not follow a rigid schema, but typically holds data in a key-value pair format.

```
public interface DocumentStoreListener {  
    void onDocumentAdded(Document document);  
    void onDocumentUpdated(Document document);  
    void onDocumentRemoved(String documentId);  
}  
  
public class RealTimeDocumentStore extends DocumentStore {  
    private List<DocumentStoreListener> listeners = new ArrayList<>();  
  
    public void addListener(DocumentStoreListener listener) {  
        listeners.add(listener);  
    }  
  
    @Override  
    public void storeDocument(Document document) {  
        super.storeDocument(document);  
        notifyDocumentAdded(document);  
    }  
  
    @Override  
    public void removeDocument(String id) {  
        super.removeDocument(id);  
        notifyDocumentRemoved(id);  
    }  
  
    private void notifyDocumentAdded(Document document) {  
        for (DocumentStoreListener listener : listeners) {  
            listener.onDocumentAdded(document);  
        }  
    }  
  
    private void notifyDocumentRemoved(String documentId) {  
        for (DocumentStoreListener listener : listeners) {  
            listener.onDocumentRemoved(documentId);  
        }  
    }  
}
```

### 2. Proposed Approach:

The Document Store Project aims to create a robust, secure, and efficient Java-based application for managing student records and important academic documents in educational institutions. To achieve this, the following approach will be implemented:

#### 1) System Design and Architecture

The system will follow a three-tier architecture to ensure separation of concerns and scalability:

- **Presentation Layer (Frontend):** A user-friendly graphical interface will be developed for faculty members to interact with the system. This will be a Java-based desktop application (or a web-based interface using Java frameworks like Spring MVC) that allows faculty to search, upload, and access student records. The interface will be intuitive and simple to navigate, ensuring quick access to essential data.
- **Business Logic Layer (Backend):** This layer will handle the core logic of the application, including file management, authentication, and authorization. All the operations related to data validation, file organization, access control, and record management will be handled here. Java's object-oriented programming (OOP) principles will be used to ensure that the application is modular and easy to maintain.
- **Data Layer (Database):** The data will be stored in a relational database such as MySQL or PostgreSQL, ensuring structured and efficient data retrieval. The database will contain tables for student information, document metadata, user roles, and access control logs. Java Database Connectivity (JDBC) or Hibernate ORM (Object-Relational Mapping) will be used to connect the application to the database.

#### 2) Document Storage and Organization

The system will store different types of student data, including academic records, personal information, certificates, and other important documents. To ensure efficient retrieval, the data will be organized into well-defined categories:

- **Student Records:** Each student will have a unique profile where their personal information (name, student ID, contact details) and academic performance (transcripts, grades, reports) are stored.
- **Document Metadata:** Each document will have associated metadata (e.g., document type, upload date, uploader's ID) that will facilitate quick searches and retrieval.

- **Folder Structure:** A hierarchical folder structure will be used to group documents by academic year, course, department, etc. This will help in better organization and navigation through student records.

### 3) Authentication and Access Control

Security and privacy of student data are critical concerns. To address this, the system will implement a role-based authentication and access control mechanism:

- **User Roles:** The system will define different roles such as Admin, Faculty, and Staff, each with varying levels of access to the data. Admins will have full access to manage user roles, while faculty will have limited access to specific records relevant to their course.
- **Authentication:** Secure authentication will be implemented using password encryption (such as hashing algorithms like crypt or PBKDF2) to ensure that users' login credentials are safely stored and protected from unauthorized access.
- **Authorization:** Each user will have access rights based on their role. Role-based access control (RBAC) will be implemented to ensure that users can only access or modify data that is within their authorization level. For example, faculty members will only be able to view or update records of students in their assigned courses, while administrators can manage the entire database.

### 4) Search and Retrieval Functionality

To enhance usability, the system will include a robust search engine that allows users to quickly find student records and documents:

- **Search by Criteria:** Faculty members will be able to search for students by various parameters such as student ID, name, course, department, or year of enrolment.
- **Advanced Search:** In addition to basic search options, advanced filters will allow users to search documents based on metadata, such as document type (e.g., transcripts, certificates) or date of upload.
- **Pagination and Sorting:** For large datasets, the system will implement pagination and sorting mechanisms to display search results in a manageable and logical manner, improving both performance and user experience.

### 5) Data Backup and Recovery

To safeguard against data loss due to system failure or human error, the system will include a backup and recovery plan:

- **Automated Backups:** The system will automatically back up the database and files at regular intervals (e.g., daily or weekly) to an external location or cloud storage.
- **Data Recovery:** In case of data corruption or system failure, the backup files can be restored to ensure minimal disruption to operations.

### 6) Scalability and Performance Optimization

The system will be designed with scalability in mind to accommodate the growth of the institution and increasing data volumes:

- **Optimized Database Queries:** Efficient queries and indexing strategies will be used to ensure fast data retrieval even with large datasets.
- **File Compression:** Documents will be stored in a compressed format to save storage space and reduce file access times.

### 7) Testing and Quality Assurance

A comprehensive testing plan will be implemented to ensure the reliability and security of the system:

- **Unit Testing:** Individual modules and functions will be tested to ensure that they perform as expected.
- **Integration Testing:** The interaction between different components (e.g., database, frontend, backend) will be tested to detect any issues.

### 8) Deployment and Maintenance

Once development and testing are complete, the system will be deployed within the educational institution's infrastructure:

- **Deployment Plan:** The system will be installed on the institution's servers, and faculty members will be trained on how to use it effectively.
- **Post-Deployment Support:** Regular maintenance and updates will be provided to fix any bugs, improve functionality, and ensure the system continues to meet the needs of the institution as it evolves.

### 9) Technological Stack

The Document Store Project will be fully Java-based, leveraging the following technologies:

- **Programming Language:** Java (with JavaFX or Spring MVC for the frontend interface)
- **Database:** MySQL or PostgreSQL (with JDBC or Hibernate for data access)
- **Authentication:** Java-based security libraries (e.g., Spring Security, crypt)
- **Storage:** File system for document storage, with metadata stored in the database

## 6. METHODOLOGY

The Document Store Project is aimed at creating a secure, efficient, and scalable Java-based system for storing, managing, and retrieving student records and academic documents. The methodology outlines the step-by-step process followed during the development, testing, and deployment of the system to ensure it meets the project's objectives. The project will adopt the Waterfall Model for its development process, which involves completing each phase of the project before moving to the next. The methodology is broken down into the following stages:

### 1) Requirement Analysis

The first step is to gather detailed requirements from stakeholders, including faculty members, administrative staff, and IT departments. This ensures that the system meets the needs of its end users while addressing key challenges in managing student data.

### 2) System Design

The design phase focuses on planning the architecture of the system, creating data models, and specifying the technologies to be used. It includes both high-level system architecture and detailed design of each component.

#### • High-Level Design:

- Three-tier Architecture: The system will be designed using a three-tier architecture:
- Presentation Layer (Frontend): Responsible for the user interface that allows faculty to interact with the system.
- Business Logic Layer (Backend): Handles the core functionality of the system, including data processing and user authentication.
- Data Layer: Manages the storage of student records and document files in a relational database.

#### • Detailed Design:

- Database Schema: Design the database schema to store student information, document metadata, user roles, and access logs. Tables will be created for students, documents, roles, and access control.
- Data Flow Diagrams: Illustrate how data flows between the frontend, backend, and database to ensure smooth communication between system components.
- User Interface (UI) Design: Mock-ups and wireframes of the frontend interface will be developed to visualize how the system will look and function for users.

### 3) Technology Selection

As the project is fully based on Java technology, appropriate tools and frameworks will be selected for different parts of the system:

- **Programming Language:** Java will be used for both frontend and backend development due to its robustness and security features.
- **Frameworks:**
  - Spring Boot for the backend to handle data processing, authentication, and role-based access control.
- **Database:** MySQL will be used to store structured data, such as student records and document metadata.

### 4) Implementation

This phase involves writing the actual code to build the system based on the design documents. The system will be developed in modules to ensure each part can be independently tested and integrated later. The implementation will be divided into the following steps:

- **Frontend Development:** The user interface will be developed using JavaFX or Spring MVC, ensuring it is intuitive and easy to navigate. Faculty members will have a dashboard to upload, manage, and search for student records and documents.
- **Backend Development:** The backend will handle the core business logic, including data processing, authentication, and document management. Role-based access control (RBAC) will be implemented to ensure that only authorized users can access or modify student data.
- **Database Integration:** The database schema will be implemented, and connections will be established between the backend and the database using JDBC or Hibernate ORM. This will ensure that student records and document files are properly stored and retrieved as needed.



## 5) Testing and Validation

After the system has been implemented, thorough testing will be conducted to ensure that it meets the project's requirements and functions as expected. Various types of testing will be performed, including:

- **Unit Testing:** Individual components and modules (e.g., user login, document upload) will be tested to verify their functionality in isolation. This helps identify and resolve bugs early in the development process.
- **Integration Testing:** After unit testing, the interaction between different system components will be tested. This ensures that the frontend, backend, and database communicate properly and function together as a cohesive system.
- **Security Testing:** The authentication and access control mechanisms will be rigorously tested to ensure they protect the system from unauthorized access and potential vulnerabilities, such as SQL injection or session hijacking.

## 7. EXPECTED OUTCOMES

When developing a document store project using Java, the expected outcomes will typically revolve around efficient storage, retrieval, and management of semi-structured or unstructured documents. Here's a list of potential outcomes, broken down by functionality and technical goals:

### 1) Efficient Document Storage

- **Scalable Storage Model:** The document store should be able to handle a large number of documents (text, PDF, etc.) and scale horizontally as needed.
- **Flexible Schema:** As document stores often manage semi-structured data, the project should allow storing documents with different structures without a predefined schema.

### 2) Document Retrieval

- **Efficient Querying:** The ability to quickly retrieve documents based on various search criteria (e.g., keyword search, tag-based search, metadata queries).
- **Support for Complex Queries:** Implement filtering, sorting, and aggregation capabilities to retrieve specific subsets of documents or analyse data.

### 3) Document Management

- **CRUD Operations:** Full support for Create, Read, Update, and Delete (CRUD) operations on documents.
- **Document Relationships:** Support for relationships between documents (e.g., parent-child, references).

### 4) Security and Access Control

- **Authentication:** Implement user authentication to restrict access to the document store.
- **Authorization:** Role-based or attribute-based access control to ensure that only authorized users can perform certain actions (e.g., read, write, update).

### 5) Performance and Scalability

- **Optimized Retrieval:** Document retrieval should be optimized in terms of latency, especially as the number of stored documents grows.
- **Efficient Indexing:** Implement strategies for indexing that minimize the time and resources required for document searches.

### 6) User Interface

- **Web Interface:** For easier interaction with the document store, a simple user interface might be built. This could allow users to upload, search, edit, and manage documents via a browser.

### 7) Testing and Documentation

- **Unit and Integration Testing:** Ensure all components of the document store are thoroughly tested. Unit tests for CRUD operations, search functionality, and security measures.
- **API Documentation:** If APIs are exposed, proper documentation should be provided to allow users to integrate with the document store.

## 8. CONCLUSION

In this Document Store project, we have successfully implemented a Java-based application that simulates the functionality of a document storage system. The main objective of this project was to design and develop a reliable, scalable, and efficient document store, which allows users to store, retrieve, and manage documents such as student details and content.

### Key Achievements:

- **Document Storage and Retrieval:** We built a system that allows users to add documents to the store and efficiently retrieve them based on search criteria like document ID, title, and tags. The use of appropriate data structures ensures quick access to stored documents.

- Search and Filtering: The system supports advanced search functionality, which allows users to filter documents based on keywords and date of creation. This was implemented using Java's built-in collections and optimized searching algorithms.
- Persistence and Data Integrity: Documents are stored in a local database (e.g., MySQL) or file system, ensuring data is persisted even after the application is shut down. We ensured data consistency and integrity during CRUD (Create, Read, Update, Delete) operations.
- User Interface: The application includes a basic command-line interface (CLI) or graphical user interface (GUI), enabling users to interact with the system in a straightforward manner. The user interface is designed to be intuitive, with options for uploading, searching, and managing documents.
- Error Handling and Logging: Robust error handling was implemented to handle common issues such as missing fields, corrupted files, or database connectivity errors. The use of logging frameworks allowed for effective tracking of system behaviour and troubleshooting.

## 9. REFERENCES

- [1] Java Documentation Oracle. (2023). *The Java™ Tutorials: Official Documentation*. <https://docs.oracle.com/javase/tutorial/> This official Java documentation provides an extensive guide on Java programming, covering fundamental concepts, libraries, and APIs used in the project, such as collections, file I/O, and JDBC.
- [2] Head First Java (2nd Edition) Sierra, K., & Bates, B. (2005). *Head First Java* (2nd ed.). O'Reilly Media. This book offers a beginner-friendly approach to learning Java, explaining key concepts such as object-oriented programming, which is essential for understanding the structure of the document store system.
- [3] Java Persistence API (JPA) Documentation Oracle. (2023). *Java Persistence API (JPA) Specification*. <https://javaee.github.io/javaee-spec/javax-persistence/> This document explains how to use JPA for managing data persistence in Java applications, which is particularly useful if the document store project involves using databases for document storage.
- [4] SQLite Documentation SQLite Consortium. (2023). *SQLite Documentation*. <https://www.sqlite.org/docs.html> If your project uses SQLite for document storage, the SQLite documentation provides a detailed explanation of how to implement and query a relational database for storing documents.
- [5] Spring Framework Documentation Pivotal. (2023). *Spring Framework Documentation*. <https://docs.spring.io/spring-framework/docs/current/reference/html/> If the project uses the Spring framework for dependency injection, transaction management, or other aspects, this resource provides detailed instructions on using Spring with Java applications.
- [6] Log4j Documentation Apache. (2023). *Apache Log4j 2 User's Guide*. <https://logging.apache.org/log4j/2.x/manual/> This guide explains how to configure and use Apache Log4j for logging in Java applications, which is helpful for tracking activities and debugging the document store system.
- [7] Java File I/O (NIO) Tutorial Buxton, K. (2022). *Java NIO File I/O Tutorial*. <https://www.baeldung.com/java/java-nio>
- [8] This article on Baeldung provides an in-depth look at Java NIO (New Input/Output), which is useful for managing file storage and retrieval within the document store system.
- [9] Studies (2016): 8-13. 2. Digital India. <https://visual.lv/community/infoaraphic/business/9-pillars-diaital-india-proaramm>