Future Enhancements

Author: Dhawanit Bhatnagar

Date: 02-August-2025

Version: 1.0

Table of Contents

---

1. Introduction

This document outlines **future improvements** to make the DMS more **scalable, feature-rich, and user-friendly**.

For each enhancement, a **step-by-step approach** is provided for implementation.

---

2. Planned Enhancements

2.1 Notifications Module

Goal: Notify users (email & in-app) when:

- Ingestion is completed/failed
- Document updates occur
- Admin changes user permissions

Steps to Implement:

1. **Backend:**
    - Add a notifications table (user_id, message, type, status, timestamp).
    - Implement event emitters on document upload, ingestion success/failure, role updates.
    - Integrate **NodeMailer** or AWS SES for email alerts.
2. **Frontend:**
    - Add a notification bell icon with dropdown.
    - Use **WebSockets (Socket.IO)** or **Server-Sent Events** for real-time notifications.
3. **Testing:**
    - Simulate ingestion completion and verify notifications reach correct users.

2.2 Advanced Search and Filtering

Goal: Enable full-text search by title, description, file content, and tags.

Steps to Implement:

1. Use **PostgreSQL Full-Text Search** or **Elasticsearch** for indexing documents.
2. Modify backend /documents API to support:
    - Search by multiple fields
    - Filter by file type, status, uploader
3. Update frontend:
    - Add search bar and filters (dropdown for type/status).
4. Test with a large dataset (100k+ documents).

2.3 Audit Logging & Version Control

Goal: Track every document change and allow reverting to previous versions.

Steps to Implement:

1. Create a document_versions table storing:
   - Document ID, version number, file path, updated by, timestamp.
2. Modify update API:
   - Save old version in document_versions before overwriting.
3. Frontend:
   - Add “View History” and “Revert” options in document actions.
4. Test rollback scenarios and data consistency.

2.4 Bulk Document Upload

Goal: Allow multiple files to be uploaded simultaneously.

Steps to Implement:

1. Update frontend upload form:
   - Accept multiple files input.
2. Update backend /documents POST API:
   - Loop through files, store metadata and files concurrently.
3. Add progress bar for user feedback.
4. Optimize performance with **parallel uploads** and queue processing.

2.5 Scheduled Ingestion Retries (Job Queue)

Goal: Automatically retry failed ingestions after a defined interval.

Steps to Implement:

1. Integrate **Bull (Redis-based)** job queue.
2. Create retry-ingestion-queue with configurable retries (3 attempts).

3. Backend CRON job:
    - Check FAILED ingestions every X minutes, requeue them.
4. Add UI toggle for admin to enable/disable auto-retry.

2.6 Cloud Storage Enhancements (Multi-Provider)

Goal: Allow switching between **AWS S3, Azure Blob, Google Cloud Storage** dynamically.

Steps to Implement:

1. Abstract file storage service (IStorageProvider).
2. Implement S3, Azure, and GCS providers.
3. Add STORAGE_PROVIDER env variable with provider options.
4. Admin UI to change provider without code deployment.

2.7 Role-Based Access Improvements (Granular Permissions)

Goal: Introduce fine-grained permissions beyond current roles.

Steps to Implement:

1. Create permissions table (create, edit, delete, trigger ingestion, view logs).
2. Map roles → permissions dynamically.
3. Implement PermissionsGuard in NestJS for per-action checks.
4. UI for admin to toggle permissions for each role.

2.8 AI-Powered Document Summarization and Q&A

Goal: Enable users to ask questions and get AI-based answers from document content.

Steps to Implement:

1. After ingestion, send text data to **OpenAI embeddings** or a local **vector DB (Pinecone, Weaviate, PostgreSQL pgvector)**.
2. Store embeddings linked to document ID.
3. Build /ask API:

    - Search embeddings for relevant content.
    - Use GPT model to summarize or answer queries.
4. UI: Add a chat-like interface per document.

2.9 Mobile-Responsive UI / PWA Support

Goal: Make DMS mobile-friendly and installable as a Progressive Web App.

Steps to Implement:

1. Use **Responsive CSS (Tailwind)** to optimize mobile layout.
2. Add a **service worker** for offline caching.
3. Generate **PWA manifest** for app-like experience.
4. Test on multiple devices.

---

3. Implementation Roadmap

Phase	**Feature**
1	Notifications, Bulk Upload
2	Advanced Search, Audit Logging
3	Auto-Retry Ingestion, Multi-Cloud Storage
4	AI Q&A Integration
5	Mobile/PWA Support

---

4. Conclusion

These enhancements will make the Document Management System:

- More **robust** and **user-friendly**
- Capable of **handling large-scale document ingestion**
- Ready for **enterprise-level adoption** with AI features and cloud flexibility