Design Decision Document

Author: Dhawanit Bhatnagar

Date: 25-July-2025

Version: 2.0

1. Purpose

The purpose of this document is to capture all major technical decisions made during the design and development of the Document Management System. It provides detailed reasoning, trade-offs, and future considerations for each choice to ensure transparency and scalability.

2. Backend Technology Choices

We evaluated the following backend frameworks:

1. **Express.js**
   - Pros: Lightweight, flexible, widely used in Node.js ecosystem.
   - Cons: No built-in modular architecture, manual setup for large projects.

2. **Django (Python)**
   - Pros: Rapid development, batteries-included framework.
   - Cons: Different tech stack from frontend (TypeScript), more overhead for real-time and microservices.

3. **NestJS (Chosen)**
   - Pros: Built with TypeScript, modular architecture, dependency injection, scalable for large teams, easy testing.
   - Cons: Slightly steeper learning curve compared to Express.

✅ **Decision:** We chose NestJS for its structure, scalability, and compatibility with TypeScript for full-stack consistency.

3. Frontend Technology Choices

1. **Angular**
   - Pros: Strong opinionated framework, two-way binding, enterprise-ready.

- Cons: Heavyweight, steeper learning curve, slower iteration for small teams.

2. **Vue.js**
   - Pros: Lightweight, beginner-friendly, fast rendering.
   - Cons: Smaller community than React, fewer enterprise integrations.

3. **React.js (Chosen)**
   - Pros: Large ecosystem, reusable components, easy state management with React Query, wide developer availability.
   - Cons: Requires additional libraries for routing and state management.

✅ **Decision:** React.js chosen for flexibility, large community support, and fast prototyping.

4. Database Choices

1. **MySQL**
   - Pros: Popular, reliable.
   - Cons: Limited support for JSON and complex indexing.

2. **MongoDB**
   - Pros: NoSQL, good for unstructured data.
   - Cons: Not ideal for relational, transactional operations.

3. **PostgreSQL (Chosen)**
   - Pros: ACID-compliant, supports relational integrity, indexing, and scalability.
   - Cons: Slightly more configuration required.

✅ **Decision:** PostgreSQL is chosen for structured data (Users, Documents, Ingestion Logs) and reliable transactions.

5. Storage Choices

1. **Local File Storage**
   - Pros: Quick to implement, free for development/testing.
   - Cons: Not scalable for production.

2. **AWS S3 (Chosen for production)**
   - Pros: Reliable, secure, highly scalable cloud storage.

- Cons: Requires IAM setup and cost per storage/transfer.

✅ **Decision:** Hybrid approach. Local storage for development, S3 for production.

6. Deployment Choices

- **Containerization:** Docker chosen for consistent environment and fast setup.
- **Orchestration:** Docker Compose to manage services (Frontend, Backend, PostgreSQL).
- **Auto-Restart:** Configured restart policy for high availability.

7. Scalability and Concurrency Design

- Backend is stateless, supports horizontal scaling via container replicas.
- PostgreSQL can be scaled with read replicas if needed.
- Ingestion process is asynchronous, allowing multiple documents to be processed concurrently.
- Future plan: Introduce a message queue (RabbitMQ or Kafka) for ingestion tasks to decouple processing.

8. Security Considerations

- JWT-based authentication to secure APIs.
- Role-based access control for Admin, Editor, and Viewer.
- Input validation and sanitization to prevent injection attacks.
- CORS policy configured for frontend-backend communication.
- Future enhancement: HTTPS enforcement, security headers, and rate-limiting.

9. Future Architectural Enhancements

- Split ingestion into a Python microservice for AI-based document processing (RAG pipeline).
- Add a Notification Service using WebSockets or SNS to inform users of ingestion status updates.
- Implement document versioning and audit logs for compliance.

- Scale with Kubernetes and load balancing for high traffic scenarios.

This document provides a clear understanding of the technical decisions made to ensure system scalability, maintainability, and future-proofing.