# Technical Specification Document (TSD)

Project Name: DocuGenAI – AI-Powered Intelligent Document Management System

## 1. Introduction

Objective:  
To develop an open-source, full-stack, AI-powered document processing platform where users can upload various types of documents and extract useful information (text, summaries, answers to questions, classifications) using Generative AI and traditional NLP, all deployed using modern DevOps practices.

Scope:  
- File upload via web UI  
- Store in MinIO (object storage)  
- OCR + GenAI processing (summarization, Q&A, classification)  
- Store results in MongoDB  
- Dashboard for visual insights  
- DevOps: Docker, Jenkins, Kubernetes, Ansible, Terraform  
- 100% Open Source Stack

## 2. SDLC Phases Overview

1. Requirements Gathering - Functional + Non-functional requirements  
2. System Design - Architecture, data flow, tech stack  
3. Development - Modular implementation plan  
4. Testing - Unit, integration, CI pipeline  
5. Deployment - Containerized deploy to Kubernetes  
6. Maintenance - Logs, monitoring, updates

## 3. Functional Requirements

* FR1: User can upload PDF, image, or Word documents
* FR2: Backend stores file in MinIO and metadata in MongoDB
* FR3: OCR extracts text using Tesseract
* FR4: AI microservice summarizes and classifies content
* FR5: User can ask questions about document content
* FR6: Dashboard shows summary, metadata, and answers
* FR7: Admin can monitor processing status

## 4. Non-Functional Requirements

* Performance: Response time < 3s for OCR + summarization
* Scalability: Deployable to Kubernetes, scale AI microservice
* Availability: 99% uptime on local or cloud infra
* Portability: All components Dockerized
* Security: File input validation, secure API endpoints

## 5. System Architecture

High-Level Architecture:  
[ React Frontend ]  
 ↓  
[ Node.js API ]  
 ↓  
[ MinIO + MongoDB ]  
 ↓  
[ AI Microservice (FastAPI + HuggingFace) ]  
 ↓  
[ DevOps Stack: Docker, Jenkins, Terraform, K8s ]

## 6. Technology Stack

* Frontend: React.js, Tailwind/MUI
* Backend: Node.js + Express
* Storage: MinIO (Object), MongoDB (NoSQL)
* AI/ML: Python, FastAPI, HuggingFace Transformers, Tesseract
* DevOps: Docker, Jenkins, Ansible, Terraform
* Orchestration: Kubernetes (Minikube/K3s/EKS)

## 7. Folder Structure

docugenai/  
├── frontend/ # React App  
├── backend/ # Node.js + Express  
├── ai-service/ # FastAPI + Transformers  
├── k8s/ # Kubernetes Manifests  
├── terraform/ # Infra setup (EC2, MinIO)  
├── ansible/ # Playbooks for provisioning  
├── Jenkinsfile # CI/CD pipeline  
└── README.md

## 8. Testing Plan

* Unit Tests: Jest, Mocha - For backend and ML endpoints
* Integration: Postman, Supertest - API route testing
* AI Tests: Pytest - Validate OCR and summarization accuracy
* CI Pipeline: Jenkins - Automated test triggers
* Manual UI Tests: Browser - Upload, dashboard validation

## 9. Deployment Strategy

* Local Dev: Docker Compose
* CI/CD: Jenkins, GitHub Actions
* Infra Provisioning: Terraform (EC2, MinIO, MongoDB)
* Provisioning: Ansible for Docker/K8s setup
* K8s Cluster: Minikube locally or AWS EKS
* Monitoring: Prometheus + Grafana (optional)

## 10. Project Timeline (10 Weeks)

1. Week 1: Project setup, GitHub repo, initial UI/backend
2. Week 2: File upload + MinIO integration
3. Week 3: OCR + MongoDB integration
4. Week 4: AI summarization/classification
5. Week 5: Question-answering and embeddings
6. Week 6: Full frontend dashboard
7. Week 7: Dockerize all services
8. Week 8: Jenkins + CI/CD pipelines
9. Week 9: Deploy with K8s using Terraform + Ansible
10. Week 10: Final testing, polish, documentation, demo recording

## 11. KPIs (for Demo & Resume)

* Document upload success rate: 100%
* Summarization speed: < 3s for 1-page doc
* Q&A accuracy: > 85% with clean text
* CI/CD success: Green builds with test pass
* Infra spin-up: < 5 mins with Terraform
* Interview Readiness: GitHub repo + demo video + resume bullet ✅