Technical Assessment Document

Contract Management System – Approach & Requirements

Author: Aishwarya Kumar Arvind

Date: [Insert Date]

# 1. Understanding of the Problem

The current contract management process relies on manual Word templates stored in shared drives, which causes inefficiencies including difficulty locating contracts, inconsistent version control, lack of lifecycle visibility, manual tracking of renewals, and delays in customer service response times.  
  
Goal: Build a secure, scalable, and user-friendly system to digitize contract storage, enforce lifecycle management, and provide fast search/filtering — improving efficiency and customer service.

# 2. Background Study

## 2.1 Industry Research

I reviewed how leading contract lifecycle management (CLM) tools operate, including Ironclad, Agiloft, OneFlow, and Fynk.  
  
- Ironclad – Focuses heavily on workflow automation, clause libraries, and legal operations dashboards.  
- Agiloft – Known for flexibility and deep customization; strong workflow rules and compliance tracking.  
- OneFlow – Emphasizes real-time collaboration on contracts with a lightweight interface.  
- Fynk – Targets startups/SMBs with a focus on templates, fast setup, and cloud-native usability.  
  
Observation: All systems prioritize centralized contract storage, metadata-driven search, lifecycle visibility, auditability, and increasingly AI/automation.

## 2.2 Contract Lifecycle Management Concepts

Industry-standard contract lifecycle stages include:  
- Drafting  
- Review/Negotiation  
- Approval/Execution  
- Active Management  
- Amendments/Renewals  
- Expiry/Archival  
  
For this assessment, scope will be Draft, Active, Expired, Renewed — with room to expand.

## 2.3 Application to This Assessment

The study reinforced that metadata, lifecycle tracking, and searchability are the backbone of CLM systems. This POC will:  
- Focus on core pain points (storage, metadata, lifecycle).  
- Be architected for scalability to enterprise-grade capabilities.  
- Include AI hooks for summarization and metadata extraction.

# 3. Requirements

Functional:  
- Upload and organize contracts digitally.  
- Track metadata (client, status, type, dates).  
- Provide search and filter capabilities.  
- Implement lifecycle states (Draft, Active, Expired, Renewed).  
- Ensure data validation and error handling.  
- Build a simple, intuitive interface.  
- Include logging and monitoring.  
- Demonstrate AI tool usage.  
  
Non-Functional:  
- Secure and confidential (encryption, RBAC).  
- Scalable (storage, DB, search decoupled).  
- Cost-efficient (open-source, free tiers).

# 4. Proposed Technology Stack

|  |  |  |
| --- | --- | --- |
| Layer | Technology | Reasoning |
| Frontend | Next.js + TailwindCSS | Modern, fast, SSR-ready; good DX |
| Backend API | FastAPI (Python) | High-performance async APIs; ML-friendly |
| Database | PostgreSQL | Relational consistency, scalability |
| Document Storage | MinIO (local) / Amazon S3 (future) | Scalable, cost-effective |
| Search | Postgres full-text → Elasticsearch | Simple start, scalable upgrade path |
| AI/ML | Hugging Face, spaCy | Summarization, NER for metadata extraction |
| Auth/Security | JWT / OAuth2 | Enterprise-grade, extensible |
| Logging/Monitoring | Serilog, OpenTelemetry, Grafana | Observability and audit |
| Deployment | Docker Compose → Kubernetes | Local POC, future production scale |

# 5. System Design

The architecture consists of:  
  
- Frontend: Next.js + TailwindCSS for responsive UI.  
- Backend API: FastAPI to handle uploads, metadata, lifecycle, AI integration.  
- Storage: MinIO/S3 for contracts, PostgreSQL for metadata, optional Elasticsearch for search.  
- AI Services: Summarization and NER microservices.  
- Logging/Monitoring: Structured logs and metrics with audit trails.  
  
Data Flow:  
- Upload: Contract uploaded → validated → stored → metadata in DB.  
- Search: Query against DB or search index.  
- Lifecycle: Enforce state transitions; log every change.  
- AI: Extract metadata or summaries for user confirmation.  
  
Scalability Considerations:  
- Stateless backend can be scaled horizontally.  
- Object storage scales independently of metadata DB.  
- Elasticsearch (future) supports high-volume full-text search.  
- Role-based access and encryption ensure confidentiality.

# 6. Timeline & Deliverables (5 Days)

- Day 1: Setup environment (Docker, repos, services).  
- Day 2: Upload + storage + metadata entry.  
- Day 3: Search/filter APIs + UI integration.  
- Day 4: Lifecycle states + logging.  
- Day 5: Optional AI integration + documentation.  
  
Deliverables:  
- Working solution.  
- Documentation of AI usage.  
- Explanation of customer service improvements.

# 7. Clarifying Questions

- Do you require role-based permissions, or is a single-user system sufficient?  
- Should metadata be entirely user-provided, or should auto-extraction be included?  
- Should AI integration be fully implemented or just demonstrated as a design hook?  
- Is local Dockerized deployment acceptable, or should it be cloud-deployed?  
- What scale (number of contracts/users) should this POC handle?  
- Should renewals trigger notifications in this version?

# 8. Conclusion

This design combines industry best practices (Ironclad, Agiloft, OneFlow, Fynk) with a modern, cost-conscious technology stack. The system addresses the immediate business pain points while remaining scalable and AI-ready. It demonstrates both practical implementation and forward-looking architecture suitable for enterprise use.