AI-Powered Virtual Legal Assistant - Project Summary

Overall Objective and Scope

The AI-Powered Virtual Legal Assistant (KanunAI) is a comprehensive legal document analysis system designed to assist legal professionals and researchers in processing, understanding, and extracting insights from legal documents. The system addresses the critical challenge of efficiently analyzing voluminous legal documents, including case judgments, contracts, statutes, and legal precedents.

Primary Objectives:

- Automate legal document summarization and analysis
- Enable semantic search and retrieval of legal information
- Provide interactive Q&A capabilities grounded in document content
- Extract structured information (timelines, key clauses, risks)
- Identify relevant legal precedents based on case similarity

Scope:

- Support for PDF legal documents (case judgments, contracts, legal briefs)
- Multi-document analysis with session-based context management
- Real-time interactive chatbot for document-specific queries
- Comprehensive contract analysis with risk assessment
- Timeline extraction and visualization from legal documents
- Precedent search using semantic similarity

Key Technologies Used

AI/ML Stack

- **Embedding Model:** sentence-transformers/all-MiniLM-L6-v2 (local, 384-dimensional vectors)
- **LLM:** Google Gemini 2.5 Flash (for text generation and summarization)
- **Vector Database:** FAISS (Facebook AI Similarity Search) for efficient similarity search
- **Framework:** LangChain for RAG pipeline orchestration

Backend Technologies

- **Runtime:** Node.js with TypeScript
- **Framework:** Express.js
- **Database:** MongoDB (for user management and session storage)
- **File Processing:** Multer for document uploads
- **Security:** JWT authentication, Helmet, CORS, rate limiting

Frontend Technologies

- **Framework:** Next.js 15 (React 19)
- **Language:** TypeScript
- **UI Components:** shadcn/ui, Radix UI

- **State Management:** Redux Toolkit
- **Styling:** Tailwind CSS
- **Visualization:** vis-timeline for timeline display, Mermaid for diagrams

Python AI Service

- **PDF Processing:** PyPDFLoader (LangChain)
- **Text Processing:** RecursiveCharacterTextSplitter
- **Caching:** Pickle for document and vector store persistence
- **API Integration:** google-generativeai SDK

System Working and Workflow

Core RAG Workflow

1. **Document Ingestion:**
    - User uploads PDF document via web interface
    - Backend receives file and stores temporarily
    - Python service loads PDF using PyPDFLoader
    - Document is split into pages and cached
2. **Data Preprocessing and Chunking:**
    - Documents are chunked into manageable segments (25 pages per chunk for cases, 2 pages for contracts)
    - Chunks preserve page metadata for traceability
    - Chunking strategy balances context preservation with processing efficiency
3. **Embedding Generation:**
    - Each chunk is passed through local embedding model (all-MiniLM-L6-v2)
    - Generates 384-dimensional vector representations
    - Embeddings capture semantic meaning, enabling similarity search beyond keyword matching
    - Process runs locally (no API costs, instant processing)
4. **Vector Database Indexing:**
    - Embeddings stored in FAISS index with metadata mapping
    - FAISS enables fast approximate nearest neighbor search
    - Index persisted to disk for session-based reuse
    - Supports efficient retrieval even with thousands of chunks
5. **Query Processing and Contextual Grounding:**
    - User query converted to embedding vector
    - FAISS performs similarity search (cosine similarity)
    - Top-K most relevant chunks retrieved (default K=3)
    - Retrieved chunks provide context for answer generation
6. **LLM-based Answer Generation:**

    - Retrieved context + user query sent to Gemini 2.5 Flash
    - LLM generates answer grounded in retrieved legal context
    - Response includes source citations (page numbers)
    - Reduces hallucination by constraining to document content

Feature-Specific Workflows

Case Analysis:

- Document → Chunking → Embedding → Summarization (hierarchical)
- Generates executive summary and detailed chunk summaries
- Enables Q&A with case-specific context

Contract Analysis:

- Document → Chunking → Per-chunk analysis → Aggregation
- Extracts 16 categories: parties, clauses, financial terms, risks, obligations, etc.
- Generates executive summary and comprehensive report
- Risk scoring and recommendation generation

Timeline Extraction:

- Document → Date extraction (regex patterns) → Event classification
- Generates chronological timeline with event types (filing, hearing, judgment, etc.)
- Visual timeline display in frontend

Precedent Search:

- Case summary → LLM-based similarity matching
- Searches across court types (Supreme Court, High Courts, District Courts)
- Returns top 5 similar cases with similarity reasoning

Methodology Adopted

RAG Architecture Selection

- **Why RAG:** Legal AI requires accuracy, traceability, and reduced hallucination. RAG ensures answers are grounded in actual document content, not just model knowledge.
- **Local Embeddings:** Cost-effective, privacy-preserving, and scalable. No API rate limits for embedding generation.
- **Hybrid Approach:** Local embeddings + cloud LLM balances cost, performance, and accuracy.

Chunking Strategy

- **Case Documents:** 25 pages per chunk (preserves legal argument context)
- **Contracts:** 2 pages per chunk (enables detailed clause-level analysis)
- **Metadata Preservation:** Page numbers and chunk indices maintained for source citation

Caching Strategy

- **Multi-level Caching:** Document pages, chunks, vector stores, and summaries cached

- **Session-based:** Each document session maintains its own cache directory
- **Performance:** First load 30-60s, subsequent loads <1s

Error Handling and Resilience

- **Rate Limiting:** API rate limiter prevents Gemini quota exhaustion
- **Timeout Protection:** 300-second timeout for long-running analyses
- **Graceful Degradation:** Fallback mechanisms for missing API keys
- **Output Validation:** JSON parsing with error recovery

Assumptions and Design Choices

Assumptions

1. **Document Format:** Primary focus on PDF documents (most common legal format)
2. **Language:** English-language legal documents (Indian legal system context)
3. **Document Size:** Typical documents 10-200 pages (handled efficiently)
4. **User Base:** Legal professionals, researchers, law students
5. **Network:** Stable internet connection for LLM API calls

Design Choices

1. Local Embeddings vs. API Embeddings

- **Choice:** Local embeddings (sentence-transformers)
- **Rationale:**
  - Zero cost for embedding generation
  - No rate limits
  - Privacy (sensitive legal documents stay local)
  - Instant processing (no network latency)

2. FAISS vs. Other Vector Stores

- **Choice:** FAISS (local file-based)
- **Rationale:**
  - Fast similarity search (optimized C++ implementation)
  - No external dependencies (no database server needed)
  - Efficient memory usage
  - Industry-standard (used by Facebook, Google)

3. Chunking Strategy

- **Choice:** Page-based chunking (25 pages for cases, 2 for contracts)
- **Rationale:**
  - Preserves legal document structure
  - Balances context window with processing efficiency
  - Maintains page-level traceability

4. LLM Selection

- **Choice:** Google Gemini 2.5 Flash

- **Rationale:**
  - Cost-effective for high-volume usage
  - Good performance on legal text
  - Reliable API with rate limiting support
  - Supports long context windows

5. Architecture: Microservices vs. Monolith

- **Choice:** Hybrid (separate Python AI service, integrated backend)
- **Rationale:**
  - Python ecosystem superior for ML/AI tasks
  - Node.js backend handles web requests efficiently
  - Clear separation of concerns
  - Easy to scale AI service independently

6. Frontend Framework

- **Choice:** Next.js with React
- **Rationale:**
  - Server-side rendering for performance
  - Excellent developer experience
  - Rich ecosystem (shadcn/ui, Tailwind)
  - TypeScript for type safety

7. Session Management

- **Choice:** Session-based caching with hash-based identifiers
- **Rationale:**
  - Enables multi-document analysis
  - Efficient cache reuse
  - User can switch between documents seamlessly

Technical Architecture

System Components

Frontend (Next.js):

- Document upload interface
- Analysis results display (markdown rendering)
- Interactive chatbot (Q&A interface)
- Timeline visualization
- Contract report viewer (dual-tab: full report + summary)

Backend (Express.js):

- REST API endpoints (/api/analysis/summary, /api/analysis/chat, etc.)
- File upload handling (Multer)
- Python process orchestration (spawn child processes)
- Session management

- Error handling and logging

AI Service (Python):

- Document loading and preprocessing
- Chunking and embedding generation
- Vector store creation and management
- LLM integration (summarization, Q&A, analysis)
- Timeline extraction
- Precedent search

Data Flow

1. **Upload Flow:** Frontend → Backend API → Python Service → PDF Processing → Caching
2. **Analysis Flow:** Python Service → Chunking → Embedding → Vector Store → Summarization → JSON Response → Backend → Frontend
3. **Q&A Flow:** Frontend → Backend API → Python Service → Query Embedding → FAISS Search → Context Retrieval → LLM Generation → Response → Frontend

Security Considerations

- **File Upload Validation:** File type, size limits (50MB max)
- **Temporary File Storage:** Files deleted after processing
- **API Rate Limiting:** Prevents abuse and quota exhaustion
- **Error Sanitization:** Sensitive information not exposed in error messages
- **Session Isolation:** Each document session isolated in separate cache directory

Performance Characteristics

- **Embedding Generation:** 0.1-0.5 seconds per chunk (CPU)
- **Vector Search:** <10ms for top-3 retrieval (even with 1000s of chunks)
- **First Document Load:** 30-60 seconds (embedding + summarization)
- **Subsequent Loads:** <1 second (from cache)
- **Chatbot Queries:** 2-5 seconds (retrieval + generation)
- **Contract Analysis:** 1-5 minutes (depending on document size)

Key Innovations

1. **Cost-Effective RAG:** Local embeddings eliminate embedding API costs
2. **Hierarchical Summarization:** Multi-level summaries (chunk → group → executive)
3. **Dual-Mode Analysis:** Specialized workflows for cases vs. contracts
4. **Timeline Extraction:** Automated date and event extraction from unstructured text
5. **Precedent Search:** Semantic similarity-based case matching
6. **Session-Based Context:** Multi-document analysis with context preservation

Limitations and Future Work

Current Limitations:

- PDF-only support (no Word, HTML, etc.)
- English language only
- Single-document Q&A context (no cross-document queries)
- No fine-tuning of embedding model for legal domain
- Limited to Indian legal system precedents

Future Enhancements:

- Multi-language support
- Cross-document analysis
- Fine-tuned legal embeddings
- Real-time collaboration features
- Advanced visualization (knowledge graphs, citation networks)
- Integration with legal databases (Westlaw, LexisNexis)