Legal Aid Assistant

System Architecture Documentation

Complete Technical Overview

Legal Aid Assistant - System Architecture

1. System Overview

The Legal Aid Assistant is a full-stack application combining a React frontend, FastAPI backend, and multiple AI-powered components for legal assistance. It uses RAG (Retrieval-Augmented Generation) with Pinecone vector database, conversational memory, and document generation capabilities.

2. Architecture Components

2.1 Frontend (React + Vite)

- Login/Signup Page - Firebase email/password authentication
- Home Dashboard - Legal insights, FAQ, and session management
- Chat Interface - Real-time AI conversation with message history
- Document Generator Modal - Template-based legal document creation
- Styling - Tailwind CSS with custom legal theme (Navy, Gold, White)

2.2 Backend API (FastAPI)

- POST /signup - Create new user account
- POST /login - Authenticate user with Firebase
- POST /chat - Send message to AI chatbot
- GET /templates - List available document templates
- POST /document/generate - Generate legal document PDF
- GET /session/reset - Clear conversation memory

2.3 AI Core Components

CombinedLegalChatbot (combined_chain.py)

Main chatbot orchestrator that combines memory, RAG retrieval, and LLM generation. Features intent classification to route legal queries to RAG and personal queries to memory.

MemoryChatbot (memory_chain.py)

Hybrid memory system using regex patterns for common facts (name, age, location) and LLM extraction for arbitrary facts (father name, college, etc.). Maintains conversation history and user profile.

DocumentGeneratorChain (document_chain.py)

Template-driven document generation using JSON templates and LLM. Creates professional legal documents (FIR, RTI, complaints) and exports to PDF. Uses timestamped folders to prevent overwrites.

RAG Retriever (retriever.py)

Retrieval-Augmented Generation using Pinecone vector database. Searches legal documents using semantic similarity with all-MiniLM-L6-v2 embeddings. Returns top-5 most relevant documents for legal queries.

3. Data Flow Examples

3.1 Chat Request Flow

1. User types message in chat interface
2. Frontend sends POST /chat to backend API
3. API calls CombinedLegalChatbot.generate()
4. Chatbot adds message to memory and extracts facts
5. Intent classifier determines if query is legal
6. If legal: Retriever searches Pinecone for relevant documents
7. Prompt is built with memory + context + history
8. LLM (Ollama llama2) generates response
9. Response saved to memory and returned to frontend
10. Frontend displays message in chat interface

3.2 Document Generation Flow

1. User clicks "Generate Doc" button
2. Frontend fetches templates via GET /templates
3. User selects template and fills form fields
4. Frontend sends POST /document/generate with data
5. Backend loads JSON template from src/templates/
6. DocumentGeneratorChain builds prompt with template + inputs
7. LLM generates formatted legal document text
8. FPDF library creates PDF file
9. PDF saved to timestamped folder in generated_documents/
10. Download URL returned to frontend
11. User can download the generated PDF

4. Technology Stack

4.1 Frontend Technologies

- React 18 - UI framework
- Vite - Build tool and dev server
- Tailwind CSS - Utility-first styling
- React Router - Client-side routing
- Framer Motion - Animations
- Lucide React - Icon library

4.2 Backend Technologies

- FastAPI - Modern Python web framework
- Uvicorn - ASGI server
- Firebase/Pyrebase - Authentication
- CORS Middleware - Cross-origin requests

4.3 AI/ML Stack

- Ollama - Local LLM runtime (llama2 model)
- LangChain - AI orchestration framework
- Sentence Transformers - Embedding model (all-MiniLM-L6-v2)
- Pinecone - Vector database for legal documents
- FPDF - PDF generation library

4.4 Storage

- Local JSON - User data (users.json)
- File System - Generated PDFs (generated_documents/)
- JSON Templates - Document templates (src/templates/)

5. Key Design Patterns

5.1 Intent-Based Routing

Queries are classified as legal or personal using pattern matching. RAG retrieval is only triggered for legal queries, preventing hallucination on personal questions and improving response accuracy.

5.2 Hybrid Memory System

Fast regex extraction for common facts (name, age, location, phone, email) with LLM fallback for arbitrary facts. Memory persists across conversation and can be used for document autofill.

5.3 Template-Driven Document Generation

JSON templates define document structure, fields, and instructions. LLM fills in legal language based on user inputs. Timestamped storage prevents file overwrites and maintains version history.

5.4 Stateless API with Stateful Memory

API endpoints are stateless HTTP handlers. Memory persists in backend instance during session. Can be reset via /session/reset endpoint for new conversations.

6. File Structure

```
Legal Aid for Marginal Communities - Langchain/
|
+-- app/
|   +-- api_server.py              # FastAPI backend
|
+-- src/
|   +-- combined_chain.py          # Main chatbot logic
|   +-- memory_chain.py            # Memory management
|   +-- document_chain.py          # Document generation
|   +-- retriever.py               # RAG retriever
|   +-- embeddings.py              # Embedding loader
|   +-- templates/                 # Document templates
|       +-- fir.json
|       +-- rti.json
|       +-- tenancy_complaint.json
|
+-- frontend-react/
|   +-- src/
|   |   +-- App.jsx                # Login page
|   |   +-- HomePage.jsx           # Dashboard
|   |   +-- ChatPage.jsx           # Chat interface
|   |   +-- index.css              # Global styles
|   |   +-- main.jsx               # Entry point
|   +-- tailwind.config.js         # Tailwind config
|   +-- package.json               # Dependencies
|
+-- generated_documents/           # PDF storage
+-- users.json                     # User database
+-- .env                           # Environment variables
```

7. Key Features

7.1 Intelligent Chat

- Context-aware responses using RAG from legal document database
- Personalized conversations with memory of user details
- Intent classification for accurate query routing
- Concise, professional legal guidance

7.2 Document Generation

- Multiple legal document templates (FIR, RTI, complaints)
- Auto-fill from user memory
- LLM-powered professional formatting
- PDF export with download capability

7.3 User Management

- Firebase authentication (email/password)
- Secure user data storage
- Session management
- Previous chat history

7.4 Professional UI

- Light theme suitable for legal context
- Clean, accessible interface
- Responsive design
- Smooth animations and transitions

8. Summary

The Legal Aid Assistant is a sophisticated AI-powered application that combines modern web technologies with advanced natural language processing. It provides accessible legal guidance through an intelligent chatbot and streamlines legal document creation through template-based generation.

The system architecture emphasizes modularity, with clear separation between frontend presentation, backend API, and AI components. This design enables easy maintenance, testing, and future enhancements.

Key strengths include the hybrid memory system for personalization, intent-based RAG routing for accurate legal information retrieval, and template-driven document generation for professional legal drafts.