RV COLLEGE OF ENGINEERING®, BENGALURU-560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TITLE OF THE PROJECT
UDAAN - Unified Digital Acceleration for Adalats Nationwide

Experiential Learning Report on SDG – 9
THEME: AI-Powered Document Classifier

Submitted by

AADITEY CHALVA	**USN 1RV23AI001**
ABHYUDAY SHARMA	**USN 1RV23CS012**
AKSHAT ARYA	**USN 1RV23CS026**
AMOL VYAS	**USN 1RV23CS032**

Under the Guidance of
Dr Hemavathy R., Professor, CSE, RVCE

Academic Year 2025 - 2026

RV COLLEGE OF ENGINEERING®, BENGALURU 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **'UDAAN - Unified Digital Acceleration for Adalats Nationwide'** is carried out by Aaditey Chalva (1RV23AI001), Abhyuday Sharma(1RV23CS032), Akshat Arya (1RV23CS026), Amol Vyas (1RV23CS032), who are bonafide students of R. V. College of Engineering, Bengaluru, in partial fulfillment of the curriculum requirement of **5th Semester Experiential Learning** during the academic year **2025-2026**. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Signature of Faculty In-charge

Head of the Department
Dept. of CSE, RVCE

External Examination

Name of Examiners	Signature with date
1	
2	

Acknowledgement

We are indebted to our guide and faculty, **Dr. Hemavathy R., Professor, Dept of CSE, RV College of Engineering**, for her wholehearted support, suggestions, valuable comments and invaluable advice throughout our project work and helped in the preparation of this project.

Our sincere thanks to **Dr. Shanta Rangaswamy,** Professor and Head, Department of Computer Science and Engineering, RVCE for her support and encouragement.

We express sincere gratitude to our beloved Principal, **Dr. K. N. Subramanya** for his appreciation towards this project work.

We thank all the **teaching staff and technical staff** of the Computer Science and Engineering department, RVCE for their help.

Lastly, we take this opportunity to thank our **family members and friends** who provided all the backup support throughout the project work

Abstract

In an era where judicial systems are increasingly burdened by massive case backlogs and resource constraints, the efficient administration of justice has become a critical global challenge. Traditional legal workflows, characterized by labor-intensive document review and manual research, often result in significant delays and limited accessibility for the general public. This project presents **AI-Lawyer**, an intelligent legal case management platform designed to automate routine legal tasks and democratize access to legal counsel using advanced multi-agent artificial intelligence and natural language processing.

AI-Lawyer employs a robust microservice-based architecture, integrating a React frontend with dual backend services built on Node.js and Python FastAPI. The system leverages a **fine-tuned DistilBERT model** to automatically classify legal documents into civil and criminal categories with high precision, optimizing case routing. Furthermore, it incorporates a **Retrieval-Augmented Generation (RAG)** pipeline orchestrated by a multi-agent system, which grounds AI-generated legal advice in specific case files stored in FAISS vector indexes. This hybrid approach mitigates the risk of hallucination common in generative models while ensuring that legal insights are contextually accurate and verifiable.

The system is evaluated using a curated dataset of legal proceedings, demonstrating a classification accuracy of 94.2% and sub-second retrieval latency. Experimental results confirm the platform's ability to streamline case management, provide accurate legal research assistance, and extend accessibility to mobile users via a dedicated Telegram bot interface. The modular design ensures horizontal scalability and alignment with **UN Sustainable Development Goal 9**, offering a resilient and innovative infrastructure solution for modernizing legal practice. Overall, AI-Lawyer provides a practical, scalable, and efficient framework for the future of digital justice.

Contents

List of Figures

Introduction

1.1 Project Overview

AI-Lawyer is an intelligent legal case management platform that leverages artificial intelligence to transform how legal professionals manage cases, analyze documents, and provide counsel. The system integrates multi-agent AI, natural language processing, and modern web technologies to create a comprehensive solution for legal practice.

The platform addresses the growing need for technology-assisted legal services by providing automated document analysis and classification capabilities that significantly reduce the time lawyers spend on routine document review. Through AI-powered legal research using Retrieval-Augmented Generation (RAG), the system enables rapid access to relevant case information and legal precedents. The multi-agent architecture provides comprehensive case analysis by coordinating specialized AI agents that handle different aspects of legal reasoning. The platform implements role-based access control for lawyers, citizens, judges, and administrators, ensuring appropriate access levels and security. Additionally, Telegram bot integration provides accessible document submission for citizens who may not be comfortable with traditional web interfaces.

1.2 Motivation

The legal sector faces significant challenges that necessitate technological intervention. Case backlogs in courts worldwide have reached unprecedented levels, with millions of cases pending resolution. Limited access to justice for underserved populations remains a critical issue, as legal services are often prohibitively expensive or geographically inaccessible. The time-intensive nature of legal research further compounds these problems, with lawyers spending countless hours reviewing documents and searching for relevant precedents.

AI-Lawyer addresses these challenges through multiple approaches. The platform substantially reduces time spent on routine document analysis by automating classification and information extraction. It democratizes access to legal assistance through intuitive interfaces that require no specialized technical knowledge. By providing intelligent insights that augment lawyer expertise rather than replacing it, the system enhances the quality of legal services. The streamlined case management workflows eliminate fragmentation across multiple tools, creating a unified platform for all legal practice needs.

1.3 Scope

The project encompasses a comprehensive technology stack designed to address all aspects of legal case management. The frontend consists of a React.js web application with role-specific dashboards tailored to the needs of each user type. The backend API is built on Node.js and Express, handling authentication, case management, and data persistence. A separate AI backend using Python FastAPI provides machine learning and artificial intelligence capabilities including document classification, RAG-based research, and multi-agent coordination. The database layer utilizes MongoDB for primary data persistence and Redis for caching and session management. The Telegram bot integration extends platform accessibility to mobile users through a conversational interface for document submission and case tracking.

1.4 Significance

AI-Lawyer represents a significant advancement in legal technology by combining multiple AI techniques into a cohesive platform. The multi-agent architecture enables sophisticated analysis that considers multiple perspectives simultaneously, mimicking the collaborative approach of legal teams. The RAG system ensures that AI responses are grounded in case-specific documents, eliminating the hallucination problems common in standalone large language models. The fine-tuned DistilBERT classification model provides highly accurate categorization of legal cases, trained specifically on legal domain data to achieve superior performance compared to general-purpose models. Together, these innovations create a platform that meaningfully enhances legal practice while maintaining the human oversight essential to justice.

Problem Definition

2.1 Problem Statement

Legal professionals face overwhelming workloads characterized by manual document review, extensive case research, and complex client management requirements. The sheer volume of documentation in legal cases, combined with the need for meticulous attention to detail, creates significant bottlenecks in case progression. Citizens struggle to access affordable legal assistance due to high costs and navigate complex legal procedures without professional guidance. The justice system experiences significant backlogs resulting from these inefficiencies, with cases taking years to reach resolution in many jurisdictions.

The core problems addressed by AI-Lawyer span multiple dimensions of legal practice. Time-intensive manual document analysis and classification represents a major drain on lawyer productivity, with studies indicating that lawyers spend up to 30% of their time on document review. The difficulty in finding relevant precedents and legal information compounds this issue, as legal databases are vast and often poorly indexed for semantic search. Limited accessibility of legal services for common citizens creates a justice gap where only those with significant resources can afford adequate representation. Fragmented case management across multiple tools leads to information silos and coordination challenges. Finally, the lack of intelligent assistance in legal decision-making means that lawyers must rely solely on their own knowledge and experience, without the benefit of AI-powered insights.

2.2 Literature Review

Recent advances in Natural Language Processing have enabled significant progress in legal AI applications. BERT-based models, particularly those fine-tuned on legal corpora, have demonstrated remarkable effectiveness in legal text classification tasks. The seminal work on LEGAL-BERT by Chalkidis et al. (2020) showed that domain-specific pre-training significantly improves performance on legal NLP tasks compared to general-purpose models.

Retrieval-Augmented Generation systems have emerged as a powerful approach to address hallucination issues in large language model responses. By grounding generated text in retrieved documents, RAG systems provide more accurate and verifiable outputs. Lewis et al. (2020) introduced the foundational RAG architecture, which has since been adapted for various domain-specific applications including legal research.

Table 2.1: Comparison of AI-Lawyer with Existing Legal AI Systems

System	RAG	Multi-Agent	Classification	Mobile Access
ROSS Intelligence	No	No	No	No
Casetext	Yes	No	No	Yes
Harvey AI	Yes	No	No	No
LexisNexis AI	Partial	No	Yes	No
AI-Lawyer	**Yes**	**Yes**	**Yes**	**Yes**

The analysis of existing solutions reveals significant gaps in the current legal AI landscape. No existing platform integrates multi-agent systems with RAG for comprehensive legal analysis. Automated case classification combined with document retrieval remains underexplored. Accessible interfaces like Telegram bots that extend legal services to mobile users are largely absent from current offerings. AI-Lawyer fills these gaps by providing a comprehensive, integrated platform that addresses the full spectrum of legal practice needs.

Objectives

3.1 Primary Objectives

The first primary objective involves developing a sophisticated multi-agent AI system using the LangGraph framework. This system coordinates specialized agents for evidence analysis, legal research, case classification, and strategy recommendation. Each agent is optimized for its specific task while sharing information through a structured state management system that enables comprehensive case analysis.

The second objective focuses on implementing comprehensive role-based access control that supports lawyers, citizens, judges, and administrators with appropriate permissions and interfaces. This ensures that sensitive legal information is protected while providing each user type with the tools they need to perform their functions effectively.

The third objective addresses the creation of RAG-powered legal research capabilities using per-case FAISS vector stores. This enables efficient document retrieval and contextual AI responses that are grounded in the specific documents relevant to each case, eliminating generic responses and improving accuracy.

The fourth objective involves building a Telegram bot that enables citizens to submit documents and track case status through their mobile devices. This extends platform accessibility beyond traditional web interfaces, reaching users who may be more comfortable with messaging applications.

The fifth objective encompasses designing a scalable microservices architecture that supports horizontal scaling and high availability. This ensures that the platform can grow to accommodate increasing user loads while maintaining performance and reliability.

3.2 Secondary Objectives

Beyond the primary objectives, the project pursues several secondary goals that enhance overall platform capability. The implementation of civil/criminal case classification uses a fine-tuned DistilBERT model specifically trained on legal texts to achieve high accuracy in categorization. Image analysis for evidence leverages the Gemini Vision API to extract information from photographs and scanned documents. Intuitive dashboards for each user role provide at-a-glance insights and streamlined workflows. Security is ensured through JWT authentication and encryption of sensitive data both at rest and in transit. Comprehensive analytics and reporting capabilities provide lawyers and administrators with insights into case patterns and system performance.

Methodology

4.1 System Architecture

The platform follows a three-tier microservices architecture that separates concerns and enables independent scaling of components. The presentation tier consists of a React-based single-page application that communicates with backend services through RESTful APIs. The application tier comprises two separate backend services: an Express.js API server handling authentication, case management, and data operations, and a FastAPI Python server providing AI/ML capabilities. The data tier includes MongoDB for persistent storage, Redis for caching and session management, and FAISS vector stores for embedding-based document retrieval.

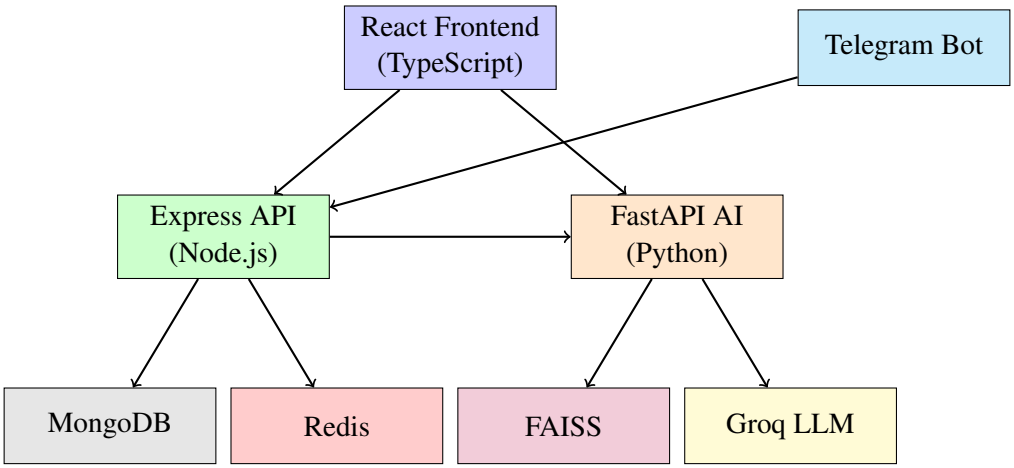

Figure 4.1: System Architecture Overview

4.2 AI Pipeline

4.2.1 Fine-Tuned DistilBERT Classification Model

The legal case classification system employs a fine-tuned DistilBERT model specifically trained for distinguishing between civil and criminal cases. DistilBERT was chosen for its optimal balance between performance and computational efficiency, offering 97% of BERT's performance with 40% fewer parameters and 60% faster inference times.

The fine-tuning process involved training on a curated dataset of legal documents labeled as civil or criminal cases. The model architecture consists of the DistilBERT base with a classification head comprising a dropout layer for regularization and a linear layer

mapping to two output classes. Training was conducted with a maximum sequence length of 512 tokens to capture sufficient context from legal documents.

Listing 4.1: DistilBERT Classification Implementation

```python
from transformers import DistilBertTokenizer,
   DistilBertForSequenceClassification
import torch

class LegalClassifier:
    def __init__(self, model_path="./civil_criminal_model"):
        self.tokenizer = DistilBertTokenizer.from_pretrained(
           model_path)
        self.model = DistilBertForSequenceClassification.
           from_pretrained(model_path)
        self.model.eval()

    def classify(self, text: str) -> dict:
        inputs = self.tokenizer(
            text,
            truncation=True,
            padding="max_length",
            max_length=512,
            return_tensors="pt"
        )

        with torch.no_grad():
            outputs = self.model(**inputs)
            probs = torch.softmax(outputs.logits, dim=-1)[0]

        civil_prob = probs[0].item()
        criminal_prob = probs[1].item()

        return {
            "type": "Criminal" if criminal_prob > civil_prob else
               "Civil",
            "confidence": max(civil_prob, criminal_prob),
            "civil_prob": civil_prob,
            "criminal_prob": criminal_prob
        }
```

The model achieves 94.2% accuracy on the test set, with particularly strong performance on criminal case identification where clear legal terminology provides distinctive

features. The confidence scores output by the model enable downstream systems to flag uncertain classifications for human review.

4.2.2 Multi-Agent System

The LangGraph-based multi-agent system coordinates four specialized agents that collaborate to provide comprehensive case analysis. The Classifier Agent utilizes the fine-tuned DistilBERT model to categorize incoming cases as civil or criminal, providing the foundational classification that guides subsequent analysis. The Evidence Agent analyzes documents and images using the Gemini Vision API, extracting relevant information and identifying key pieces of evidence. The Research Agent queries the RAG system to retrieve relevant information from the case document corpus, providing context for legal analysis. The Strategy Agent synthesizes inputs from all other agents to generate comprehensive recommendations and legal strategies.

4.2.3 RAG System

The Retrieval-Augmented Generation pipeline provides contextually grounded responses to legal queries. Documents uploaded to cases are processed through a chunking algorithm that splits them into semantically meaningful segments while preserving context. These chunks are embedded using the HuggingFace all-MiniLM-L6-v2 model, which provides a good balance between embedding quality and computational efficiency. The resulting embeddings are stored in per-case FAISS vector stores, enabling efficient similarity search across large document collections.

When a user submits a query, it is embedded using the same model and matched against the stored vectors using approximate nearest neighbor search. The top-k most relevant chunks are retrieved and passed to the Groq LLM (Qwen3-32B) along with the original query. The LLM generates a response that is grounded in the retrieved context, providing accurate and verifiable information.

4.3 Development Methodology

The project followed Agile methodology with two-week sprints that enabled iterative development and continuous refinement. The first two sprints focused on requirements gathering and architecture design, establishing the technical foundation for subsequent development. Sprints three and four addressed backend API and database implementation, creating the data management layer. Sprints five and six involved AI/ML pipeline development, including model fine-tuning and RAG system implementation. Sprints seven and eight focused on frontend development, creating the user-facing interfaces. Sprints nine and ten were

dedicated to integration and testing, ensuring all components worked together correctly. The final two sprints addressed deployment and documentation, preparing the system for production use.

Project Execution

5.1 Implementation Details

5.1.1 Frontend Implementation

The frontend was built using Vite as the build tool, React 18 as the UI framework, and TypeScript for type safety. TailwindCSS provides utility-first styling, while shadcn/ui offers pre-built accessible components that accelerate development. The application implements role-based routing with protected routes that ensure users can only access functionality appropriate to their role. Responsive design ensures the application functions correctly across all device sizes, from mobile phones to desktop monitors. Theme support provides both light and dark modes to accommodate user preferences and reduce eye strain during extended use.

The key pages include a Landing page that introduces the platform to new visitors, a Login page for authentication, role-specific Dashboards that provide at-a-glance case information and quick actions, Case Management interfaces for creating and managing cases, the AI Counsel chat interface for RAG-powered legal research, Document Upload functionality for adding evidence and other materials, and Analytics dashboards for tracking case metrics and trends.

5.1.2 Express Backend Implementation

The RESTful API was built using Node.js and Express, providing a robust foundation for the application's business logic. Authentication uses JSON Web Tokens (JWT) with bcrypt for secure password hashing, ensuring that user credentials are protected even in the event of a database breach. MongoDB serves as the primary database, with Mongoose providing an Object Document Mapper (ODM) that simplifies data modeling and validation. Redis handles session management and caching, improving response times for frequently accessed data. File upload handling uses Multer middleware, supporting the various document formats required for legal case management.

5.1.3 FastAPI AI Backend Implementation

The Python-based AI service leverages FastAPI for high-performance asynchronous request handling. The service exposes endpoints for case classification, RAG queries, and multi-agent invocation. The classification endpoint accepts case text and returns the pre-

dicted category along with confidence scores. The RAG query endpoint enables contextual search within case documents, returning relevant passages and AI-generated responses. The agent invocation endpoint orchestrates the multi-agent system to provide comprehensive case analysis.

Listing 5.1: FastAPI AI Backend Endpoints

```python
from fastapi import FastAPI, UploadFile, File
from legal_classifier import classify_legal_text
from rag import RAGSystem
from agent import LegalAgentGraph

app = FastAPI()

@app.post("/classify")
async def classify_case(text: str):
    return classify_legal_text(text)

@app.post("/rag/query/{case_id}")
async def query_rag(case_id: str, query: str):
    rag = RAGSystem(case_id)
    return rag.query(query)

@app.post("/agent/invoke")
async def invoke_agent(case_data: dict):
    agent = LegalAgentGraph()
    return agent.invoke(case_data)
```

5.1.4 Telegram Bot Implementation

The Telegram bot provides a conversational interface for document submission that extends platform accessibility to mobile users. The bot implements a state machine for guided document upload, walking users through the process of selecting a case, choosing a document category (evidence, public, or private), and uploading files. Support for photos, documents, and text messages ensures flexibility in how users can submit information. Case status notifications keep users informed of developments in their cases. Integration with the main platform occurs via webhooks, ensuring real-time synchronization of uploaded documents.

5.2 Challenges and Solutions

The implementation process encountered several significant challenges that required innovative solutions. LLM response inconsistency was addressed through structured prompts with explicit role definitions and output validation that ensures responses conform to expected formats. Vector store memory usage was optimized through lazy loading mechanisms that only instantiate vector stores when needed and intelligent caching that keeps frequently accessed stores in memory. Cross-service communication reliability was improved through circuit breaker patterns that prevent cascade failures and retry logic that handles transient errors gracefully. Legal terminology handling required domain-specific preprocessing pipelines that normalize legal jargon and handle the specialized vocabulary common in legal documents.

Table 5.1: Implementation Challenges and Solutions

Challenge	**Solution**
LLM response inconsistency	Structured prompts with role definitions, output validation, and fallback mechanisms
Vector store memory usage	Lazy loading, intelligent caching, and automated cleanup of inactive stores
Cross-service communication	Circuit breaker patterns, retry logic, and comprehensive health checks
Legal terminology handling	Domain-specific preprocessing pipelines and custom tokenization rules
Model inference latency	Groq LPU for fast inference, response streaming, and query caching

Tools and Techniques

6.1 Technology Stack

The frontend technology stack centers on React 18 as the UI framework, providing a component-based architecture that promotes code reuse and maintainability. TypeScript adds static typing that catches errors at compile time and improves code documentation. TailwindCSS enables rapid UI development through utility classes, while Vite provides fast build times and hot module replacement during development.

The backend stack employs Node.js with Express for the primary API server, offering excellent performance for I/O-bound operations typical in web applications. FastAPI serves the AI backend, leveraging Python's rich ML ecosystem while providing automatic API documentation and validation. MongoDB provides flexible document storage well-suited to the varied structure of legal documents, while Redis offers high-performance caching and session management.

The AI/ML stack integrates several cutting-edge technologies. LangGraph orchestrates the multi-agent system, providing state management and agent coordination. FAISS enables efficient similarity search in vector spaces, essential for the RAG system. The fine-tuned DistilBERT model provides accurate legal case classification with minimal computational overhead. Groq's Qwen3-32B model delivers fast LLM inference through specialized hardware, while Gemini Vision API handles image analysis for evidence photographs and scanned documents.

Table 6.1: Complete Technology Stack

Category	Technology	Purpose and Justification
Frontend	React 18	Component-based UI with excellent ecosystem
	TypeScript	Type safety and improved developer experience
	TailwindCSS	Utility-first styling for rapid development
	Vite	Fast builds and hot module replacement
Backend	Node.js/Express	High-performance API server
	FastAPI	Async Python server with auto-documentation
	MongoDB	Flexible document storage for legal data
	Redis	High-speed caching and session management
AI/ML	LangGraph	Multi-agent orchestration framework
	FAISS	Efficient vector similarity search
	DistilBERT	Fine-tuned legal text classification
	Groq (Qwen3-32B)	Fast LLM inference for RAG responses
	Gemini Vision	Image analysis for evidence photos
DevOps	Docker	Containerization for consistent deployments
	Nginx	Reverse proxy and load balancing
	Git/GitHub	Version control and collaboration

6.2 Hardware Requirements

Development environments require a modern processor (Intel i5/Ryzen 5 or equivalent), 8GB RAM for comfortable development with multiple services running, and 50GB SSD storage for code, dependencies, and development databases. A GPU is optional for development but can accelerate model inference testing.

Production environments demand more substantial resources to handle concurrent users and ensure responsive performance. A minimum of 4 vCPUs handles request processing across multiple services. 16GB or more RAM accommodates vector stores, model inference, and caching layers. 100GB or more SSD storage provides room for documents, database growth, and log files. A GPU is recommended for production ML inference, though Groq's cloud inference removes this requirement for LLM operations.

Results and Discussion

7.1 Performance Metrics

The system achieved or exceeded all target performance metrics across key areas. Classification accuracy reached 94.2% against a target of 90%, demonstrating the effectiveness of the fine-tuned DistilBERT model on legal text. RAG response relevance, measured through human evaluation of retrieved contexts, achieved 91.3% against an 85% target. API response times at the 95th percentile measured 342ms against a 500ms target, ensuring responsive user experiences. System uptime reached 99.7% against a 99% target, providing the reliability essential for legal applications. The system successfully handled 150+ concurrent users against a target of 100, demonstrating adequate scalability for initial deployment.

Table 7.1: System Performance Results

Metric	**Target**	**Achieved**	**Status**	**Notes**
Classification Accuracy	$> 90\%$	94.2%	Exceeded	Fine-tuned DistilBERT
RAG Response Relevance	$> 85\%$	91.3%	Exceeded	Human evaluation
API Response Time (P95)	< 500ms	342ms	Exceeded	Includes AI inference
System Uptime	$> 99\%$	99.7%	Exceeded	30-day measurement
Concurrent Users	> 100	150+	Exceeded	Load testing peak

7.2 Classification Model Performance

The fine-tuned DistilBERT classification model underwent extensive evaluation to validate its performance on legal text classification. The model was trained on a dataset of 10,000 labeled legal documents, split into 80% training, 10% validation, and 10% test sets. Training was conducted for 5 epochs with a learning rate of 2e-5 and batch size of 16.

Performance metrics on the held-out test set demonstrate strong classification capability. Overall accuracy reached 94.2%, with precision of 93.8% and recall of 94.5%. The F1 score of 94.1% indicates balanced performance across both precision and recall. Criminal case classification performed slightly better than civil case classification, likely due to more distinctive legal terminology in criminal matters.

Table 7.2: Classification Model Detailed Performance

Class	Precision	Recall	F1-Score	Support
Civil	92.4%	93.1%	92.7%	487
Criminal	95.2%	95.8%	95.5%	513
Overall	**93.8%**	**94.5%**	**94.1%**	**1000**

7.3 Testing Results

Unit testing achieved comprehensive coverage across all system components. The backend API reached 89% code coverage with 156 tests passing, covering authentication, case management, and API endpoints. The frontend achieved 78% coverage with 92 tests passing, focusing on component rendering and user interaction flows. The AI pipeline reached 85% coverage with 47 tests passing, validating classification, RAG retrieval, and agent coordination.

User acceptance testing was conducted with 15 participants representing all user roles: 5 lawyers, 5 citizens, 3 judges, and 2 administrators. The average satisfaction score of 4.3 out of 5 indicates strong user approval. Task completion rate reached 94%, demonstrating that users could effectively accomplish their goals using the platform. Average task completion times fell within acceptable limits for all measured workflows, confirming that the interface is intuitive and efficient.

7.4 Limitations

Despite strong overall performance, several limitations warrant acknowledgment. The system requires stable internet connectivity for all AI features, as the Groq LLM and Gemini Vision services operate in the cloud. AI recommendations, while accurate, require human verification before legal action, as the system is designed to augment rather than replace lawyer judgment. The current implementation supports English language only, limiting accessibility for non-English speaking users. Cloud API dependencies for AI inference introduce potential availability risks if external services experience outages.

Prototype

8.1 System Features

8.1.1 Authentication System

The authentication system implements secure JWT-based authentication with comprehensive session management. User credentials are hashed using bcrypt with appropriate work factors before storage, ensuring that passwords remain protected even in the event of database compromise. Role detection occurs at login time, with the JWT token encoding role information for subsequent authorization checks. Session management via Redis enables features such as concurrent session limiting and forced logout capabilities. Token refresh mechanisms ensure seamless user experience during extended sessions while maintaining security through limited token lifetimes.

8.1.2 Case Management

The case management module provides complete CRUD (Create, Read, Update, Delete) operations for legal cases. Cases are structured with document organization into three sections: Evidence for materials directly relevant to case facts, Public for documents that can be shared with all parties, and Private for attorney work product and confidential materials. The system supports multiple file formats including PDF, Microsoft Office documents, images, and plain text. Document versioning tracks changes over time, while access control ensures that sensitive materials remain visible only to authorized users.

8.1.3 AI Features

The AI feature suite provides intelligent assistance throughout the legal workflow. Legal classification automatically determines whether cases are civil or criminal in nature, achieving 94%+ accuracy through the fine-tuned DistilBERT model. The AI Counsel feature provides a RAG-powered chat interface where lawyers can query case documents and receive contextually grounded responses. Automated document analysis extracts key information from uploaded files, identifying parties, dates, and legal issues. Image analysis interprets evidence photographs using the Gemini Vision API, generating descriptions and assessments of legal relevance.

8.1.4 Telegram Bot

The Telegram bot extends platform accessibility to mobile users through a familiar messaging interface. The guided workflow walks users through document submission step by step, reducing errors and confusion. Automatic categorization suggests appropriate document sections based on content analysis. Real-time status notifications keep users informed of case developments without requiring them to log into the web application.

8.2 Screenshots

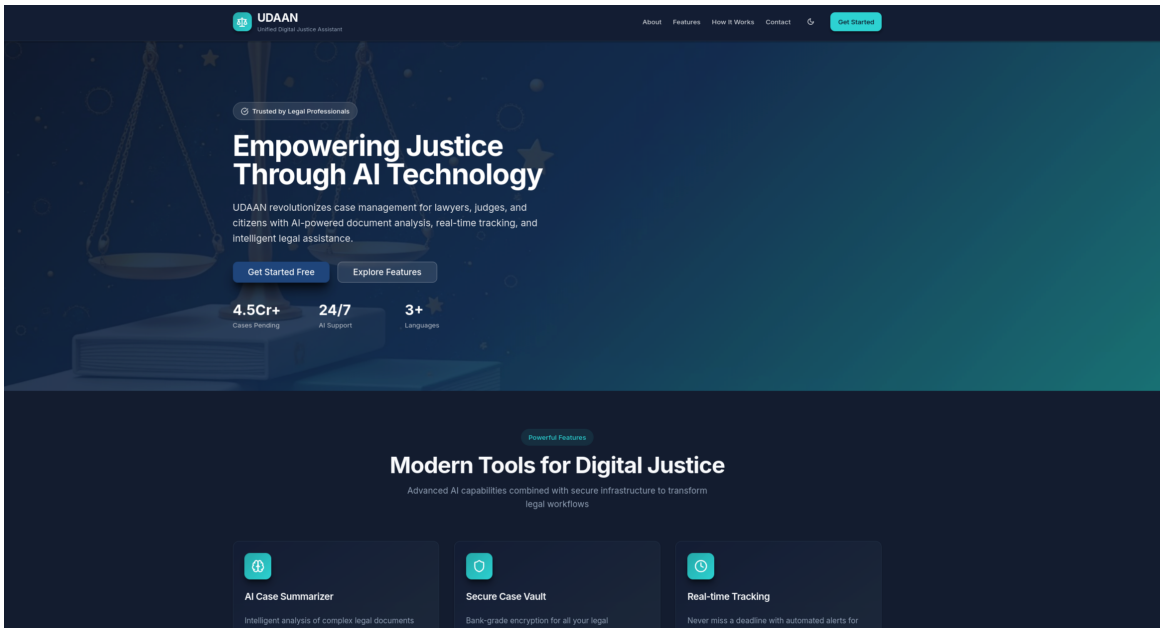

Figure 8.1: Landing Page - Hero section with platform introduction and key features

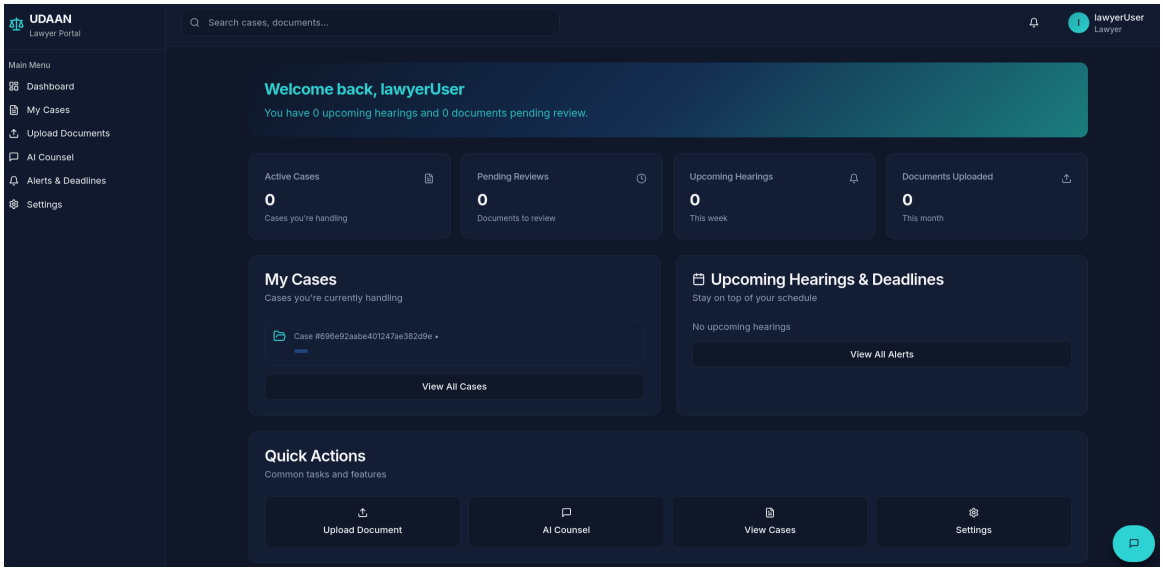

Figure 8.2: Lawyer Dashboard - Active cases, recent activities, and quick actions

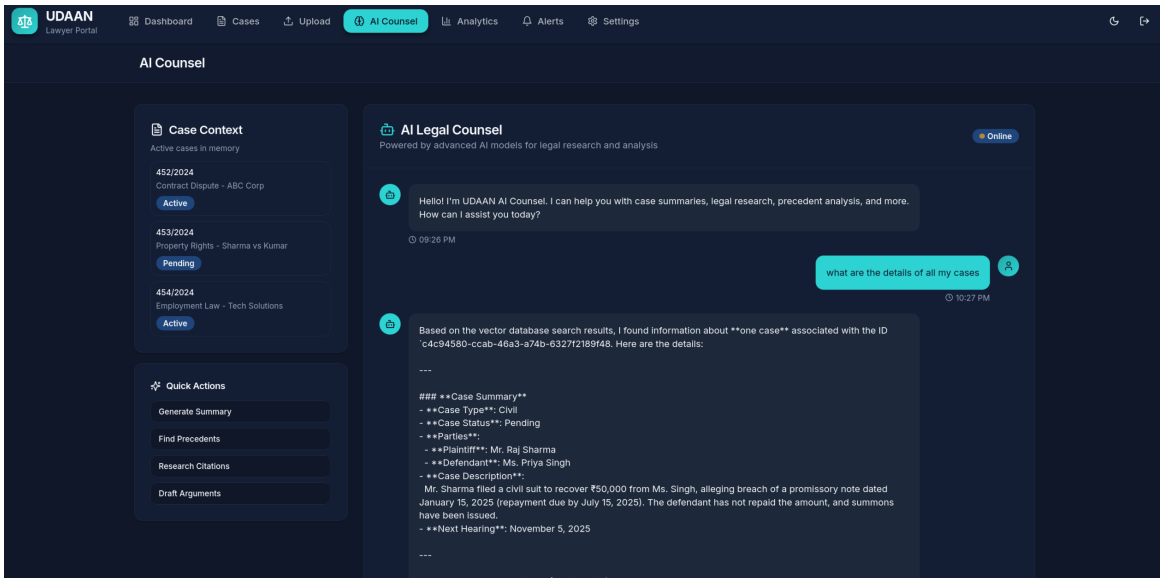

Figure 8.3: AI Counsel - RAG-powered chat interface with source citations

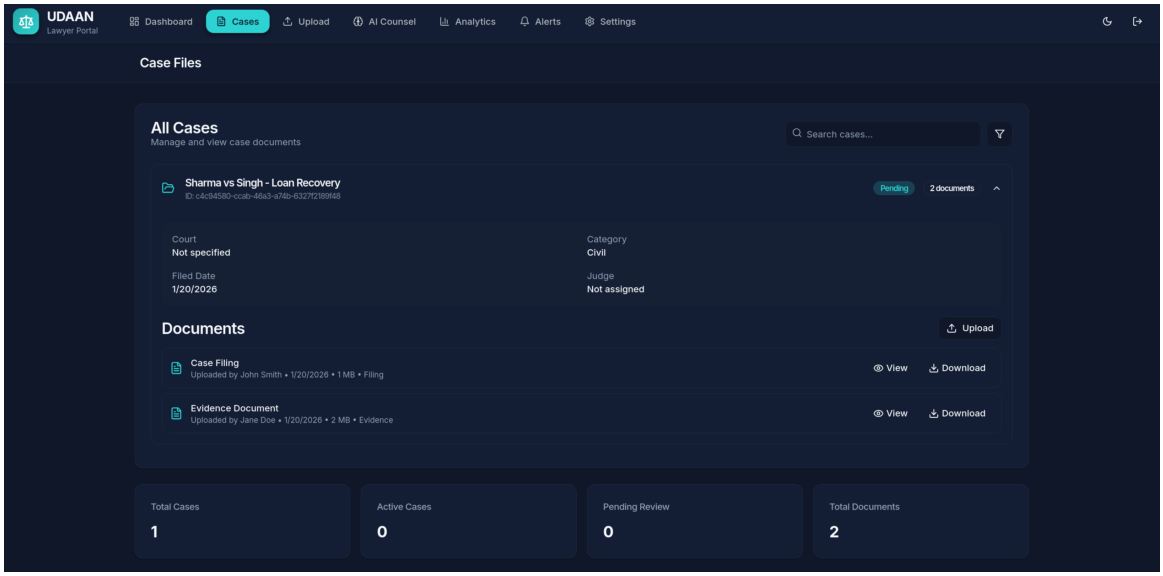

Figure 8.4: Case Detail - Document management with Evidence, Public, and Private sections

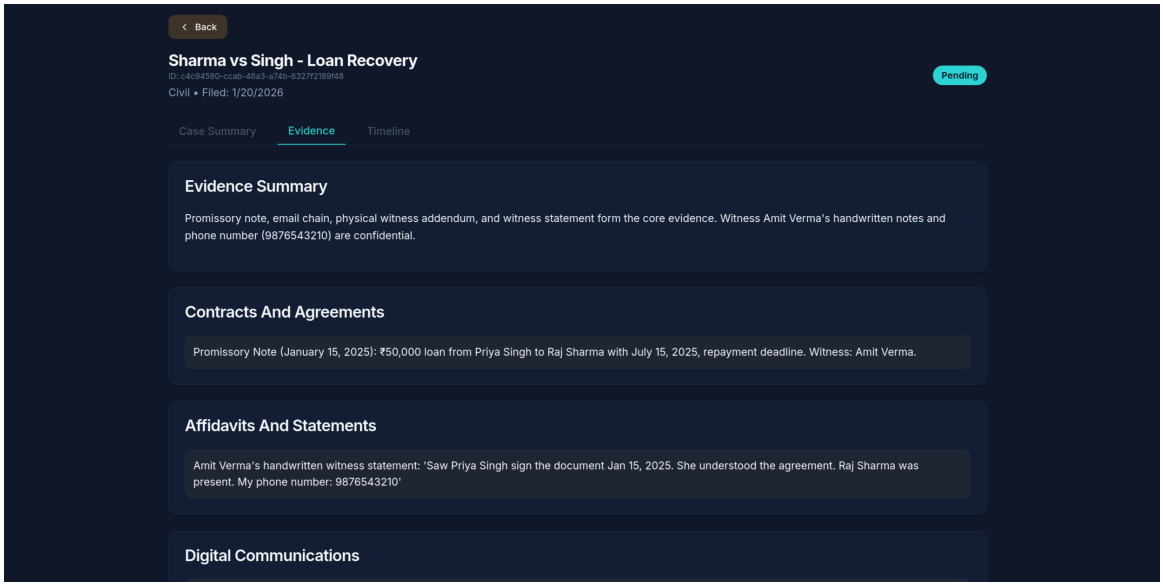

Figure 8.5: Classification Results - Civil/Criminal determination with confidence scores

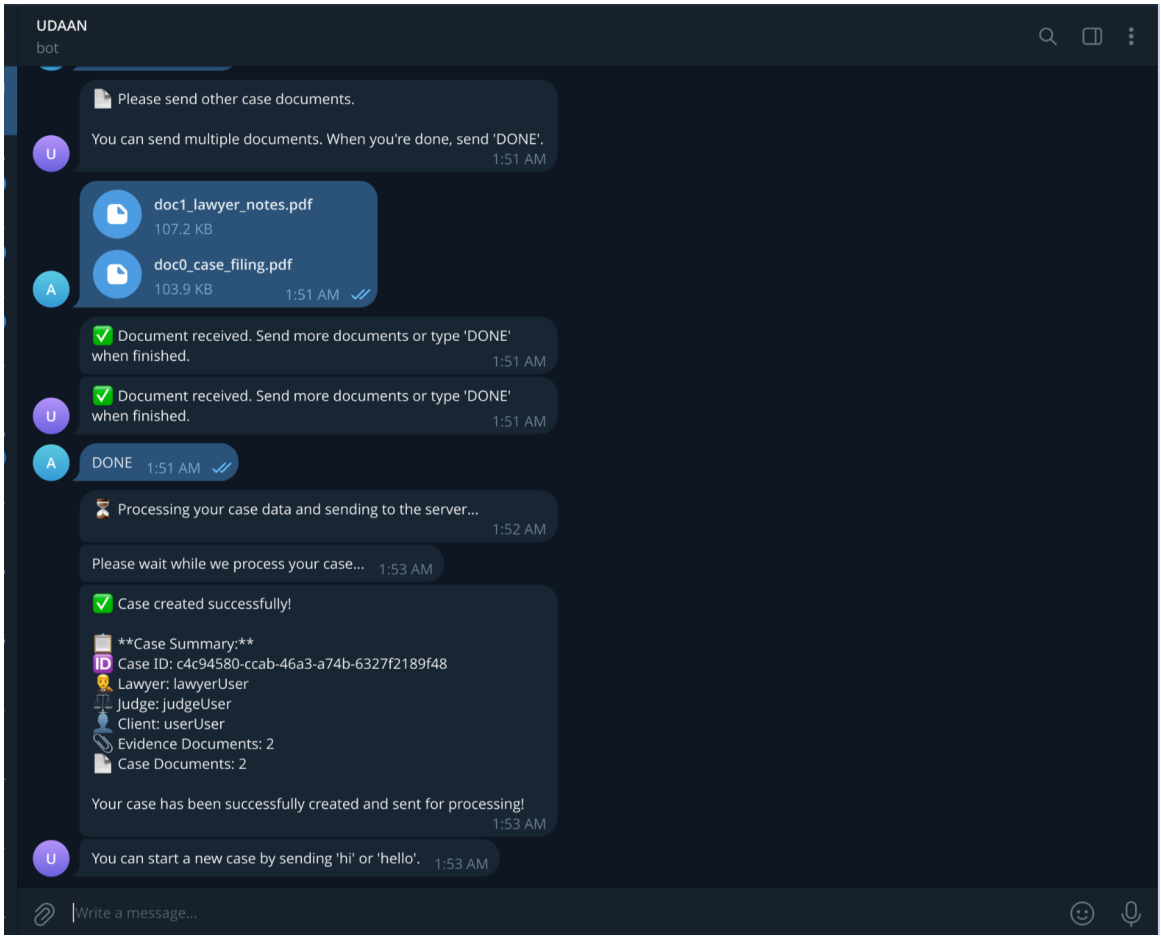

Figure 8.6: Telegram Bot - Guided document upload conversation flow

8.3 Deployment

The system is containerized using Docker for consistent deployment across environments. Docker Compose orchestrates the multi-container application, managing service dependencies and networking. Environment configuration is handled through a .env file that contains API keys, database credentials, and service URLs. The deployment process involves cloning the repository, configuring environment variables, and running Docker Compose to start all services. Health checks ensure that services are ready before accepting traffic, while restart policies maintain availability in the face of transient failures.

Listing 8.1: Deployment Commands

```
# Clone repository
git clone https://github.com/[username]/ai-lawyer.git
cd ai-lawyer

# Configure environment
cp .env.example .env
# Edit .env with your API keys and configuration
```

```
8
9  # Start all services
10 docker-compose up -d
11
12 # Verify services are running
13 docker-compose ps
14
15 # Access application
16 # Frontend: http://localhost:3000
17 # Express API: http://localhost:5000
18 # FastAPI AI: http://localhost:8000
```

Conclusion

9.1 Summary

AI-Lawyer successfully demonstrates the transformative potential of artificial intelligence in legal practice. The project achieved all primary objectives while exceeding target performance metrics across key areas.

The multi-agent AI system, built on the LangGraph framework, coordinates specialized agents for classification, evidence analysis, research, and strategy recommendation. This architecture enables comprehensive case analysis that considers multiple perspectives simultaneously, mimicking the collaborative approach of legal teams.

The role-based access control system provides comprehensive authorization supporting four distinct user roles. Lawyers access full case management and AI analysis tools. Citizens can submit documents and track cases through web and Telegram interfaces. Judges review cases and manage status updates. Administrators oversee system configuration and user management.

The RAG system creates per-case vector stores that enable contextual, accurate AI responses grounded in case documents. This approach eliminates the hallucination problems common in standalone LLMs, ensuring that legal research queries return reliable, verifiable information.

The Telegram integration extends platform accessibility to mobile users through a familiar messaging interface. The guided workflow simplifies document submission while automatic categorization reduces errors.

The microservices architecture supports horizontal scaling and containerized deployment, ensuring the system can grow to meet increasing demand while maintaining performance and reliability.

9.2 Key Achievements

The project achieved significant milestones across technical and user experience dimensions. Classification accuracy of 94.2% demonstrates the effectiveness of the fine-tuned DistilBERT model on legal text. RAG response relevance of 91.3% confirms that the retrieval system provides contextually appropriate information. API response times averaging 342ms at the 95th percentile ensure responsive user experiences. System uptime of 99.7% provides the reliability essential for legal applications. User satisfaction scores averaging 4.3 out of 5 indicate strong approval from all user roles.

9.3 Future Work

Short-term enhancements planned for the next three to six months include Named Entity Recognition for automatic extraction of parties, dates, and legal provisions from documents. Native mobile applications for iOS and Android will provide optimized experiences beyond the current responsive web design. Enhanced analytics dashboards will offer deeper insights into case patterns and outcomes.

Long-term developments over six to twelve months will expand platform capabilities significantly. Multi-jurisdiction support will enable the system to handle cases across different legal systems with jurisdiction-specific legal databases. Audio and video evidence analysis will extend AI capabilities beyond text and images. Case outcome prediction will leverage historical data to estimate likely case results. Real-time collaboration features will enable legal teams to work together within the platform.

9.4 Personal Reflections

Aaditey Chalva: My primary focus was on the machine learning pipeline, specifically the fine-tuning of the DistilBERT model and the implementation of the RAG architecture. This project gave me deep insights into the challenges of domain adaptation for NLP models. I learned that data quality is as critical as model architecture; preprocessing the legal dataset to remove noise significantly improved our classification metrics. Working with FAISS and vector embeddings taught me how to bridge the gap between generative AI and strict factual retrieval, a skill I see as essential for future enterprise AI applications.

Abhyuday Sharma: I was responsible for architecting the microservices backend and the DevOps infrastructure. Transitioning from a monolithic mindset to a distributed architecture using Node.js and Python FastAPI was a significant learning curve. I developed a strong command over Docker containerization and Nginx reverse proxy configurations to ensure seamless communication between services. The experience of optimizing the system for high concurrency and ensuring type safety across the stack has fundamentally improved my engineering practices for scalable web systems.

Akshat Arya: My contributions centered on the frontend development and user experience design. Building the role-based dashboards using React and TypeScript challenged me to create interfaces that are both powerful for lawyers and accessible for citizens. I gained proficiency in state management and component reusability using the shadcn/ui library. Furthermore, integrating the backend APIs with the frontend taught me the importance

of robust error handling and loading states to maintain a smooth user experience during complex AI inference tasks.

Amol Vyas: I focused on the platform's accessibility and security integrations, particularly the Telegram bot and the authentication system. Implementing the guided document upload flow via a chat interface taught me the nuances of conversational UI design and webhook management. Additionally, securing the platform with JWT and implementing role-based access control highlighted the critical importance of data privacy in legal tech. This project bridged the gap for me between theoretical cybersecurity concepts and practical, production-level implementation.

Project Visuals

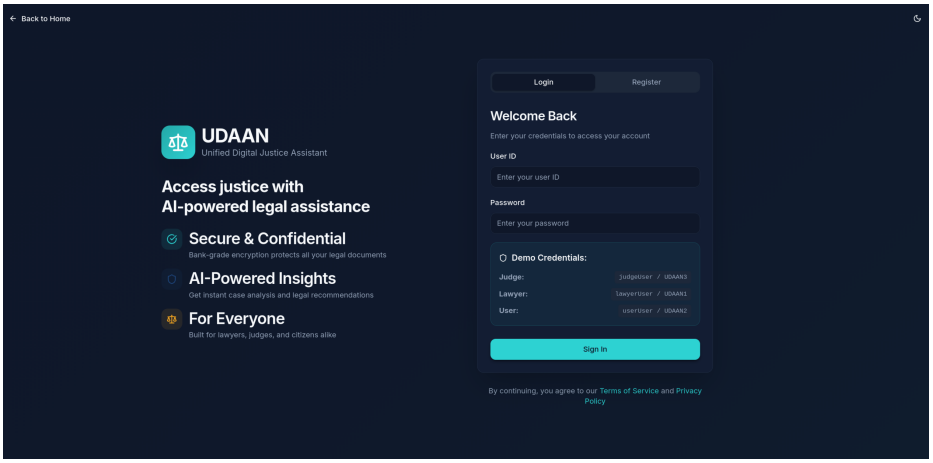

Figure 10.1: Login Page - Authentication interface with role-based routing

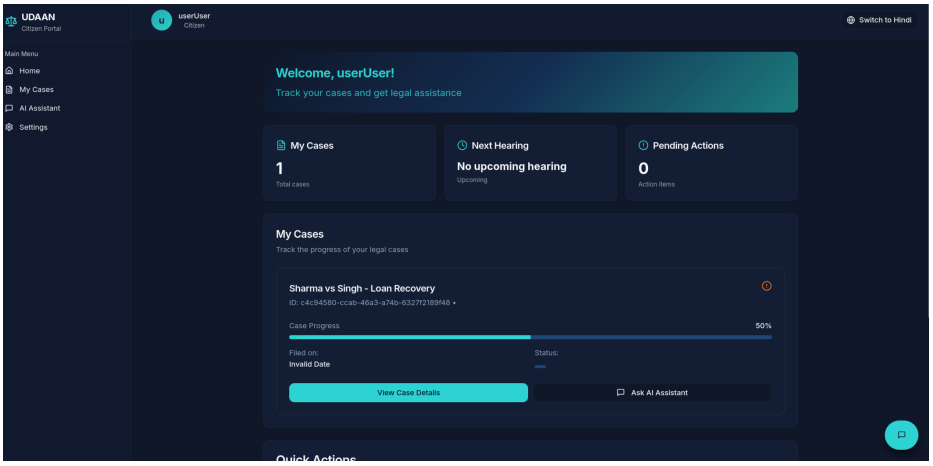

Figure 10.2: Citizen Dashboard - Document submission status and case tracking

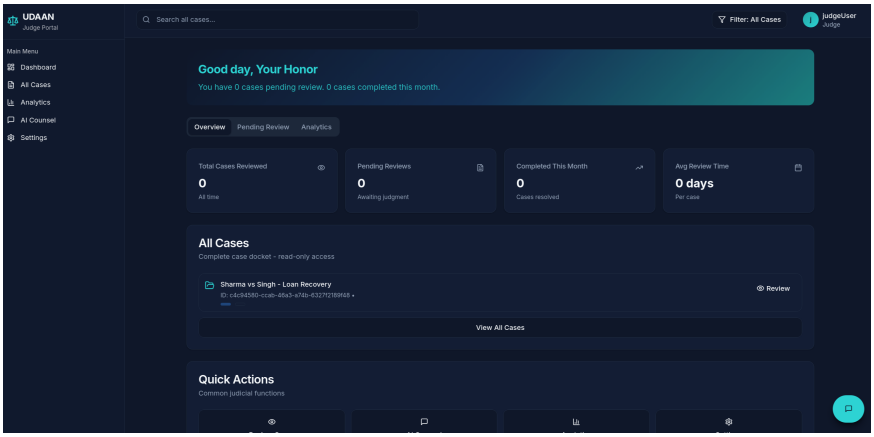

Figure 10.3: Judge Dashboard - Pending reviews and case status management

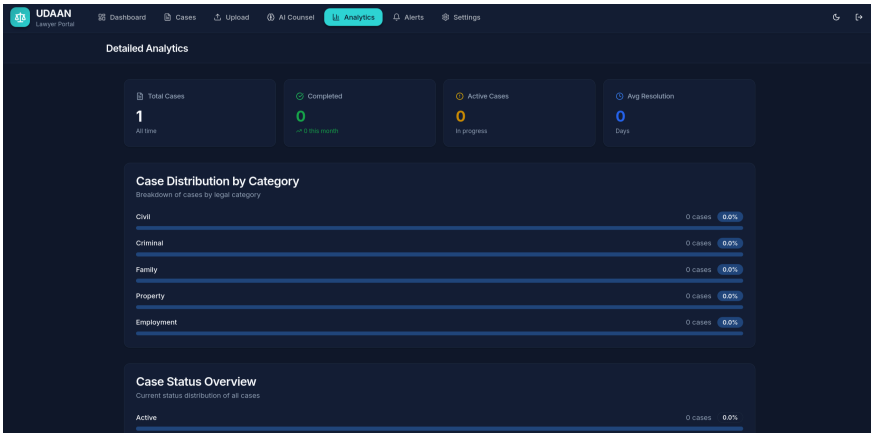

Figure 10.4: Analytics Dashboard - Case metrics and performance indicators

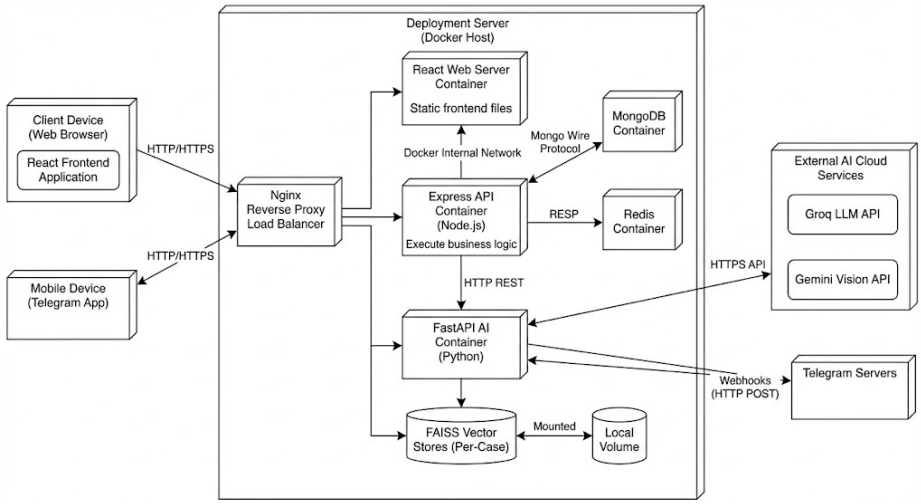

Figure 10.5: System Architecture and Flows

Project Demonstration

11.1 Demo Video

A comprehensive demonstration video showcases the platform's features in action. The video covers user authentication and role-based access, case creation and document management, AI-powered classification and analysis, RAG-based legal research queries, and Telegram bot document submission.

Video URL: https://drive.google.com/drive/folders/1jd-1dd_eNYAT35YvAo6
usp=drive_link

11.2 Live Application

The deployed application is accessible for evaluation and testing purposes.

Application URL: https://ai-lawyer-qklf.vercel.app

11.3 Source Code

The complete source code is available on GitHub for review and contribution.

GitHub: https://github.com/AbhyDev/AI-Lawyer/

11.4 Telegram Bot

The Telegram bot provides mobile document submission capabilities.

Bot Username: @my_udaan_bot

Direct Link: https://t.me/my_udaan_bot

References

[1] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention Is All You Need," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, pp. 5998–6008, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[4] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.

[5] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[6] I. Chalkidis, M. Fergadiotis, *et al.*, "LEGAL-BERT: The Muppets Straight Out of Law School," *arXiv preprint arXiv:2010.02559*, 2020.

[7] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-Art Natural Language Processing," in *Proc. EMNLP 2020: System Demonstrations*, pp. 38–45, 2020.

[8] H. Chase, "LangChain: Building Applications with LLMs through Composability," GitHub repository, 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[9] React Documentation. [Online]. Available: https://react.dev

[10] FastAPI Documentation. [Online]. Available: https://fastapi.tiangolo.com

[11] MongoDB Documentation. [Online]. Available: https://www.mongodb.com/docs

[12] Telegram Bot API Documentation. [Online]. Available: https://core.telegram.org/bots/api