

AI-Powered Legal and Finance Assistant

A Project report submitted at the end of the 4th semester in partial fulfillment of
the award of the degree of

**Bachelor of Technology in Computer Science and Systems
Engineering**

By

VENKATA SAI JASWANTH KUTIKUPPALA
(Reg.NO:321506402393)

Under the Esteemed Guidance of

Dr. I. S. SRINIVASA RAO

Professor



**DEPARTMENT OF
COMPUTER SCIENCE AND SYSTEMS ENGINEERING
ANDHRA UNIVERSITY COLLEGE OF ENGINEERING
VISAKHAPATNAM-53003**

(2021 - 2025)

**DEPARTMENT OF
COMPUTER SCIENCE AND SYSTEMS ENGINEERING
ANDHRA UNIVERSITY COLLEGE OF ENGINEERING
VISA KHAPATNAM-53003**



CERTIFICATE

This is to certify that the project report entitled "**AI-Powered Legal and Finance Assistant**", is the Bonafide work carried out by **VENKATA SAI JASWANTH KUTIKUPPALA** with Regd.No:321506402393, a student of B.Tech in **ANDHRA UNIVERSITY COLLEGE OF ENGINEERING, ANDHRA UNIVERSITY, VISA KHAPATNAM**, during the year 2021-2025, in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** (Computer Science and Systems Engineering).

Dr. I. S. SRINIVASA RAO

Project Guide

DEPARTMENT OF CS & SE
AU COLLEGE OF ENGINEERING
ANDHRA UNIVERSITY

Prof. K. Venkata Rao

Head Of the Department

DEPARTMENT OF CS & SE
AU COLLEGE OF ENGINEERING
ANDHRA UNIVERSITY

DECLARATION

I declare that the project report entitled "**AI-Powered Legal and Finance Assistant**" has been done by me in partial fulfillment of requirement for the award of degree of "**BACHELOR OF TECHNOLOGY**", during the academic year 2021-2025 under the guidance of "Dr. I. S. SRINIVASA RAO", department of Computer Science and Systems Engineering, AU College of Engineering, Andhra University, Visakhapatnam. I, here by declare that this project work has not been submitted to any other universities/institutions for the award of any degree.

PLACE: VISAKHAPATNAM

VENKATA SAI JASWANTH KUTIKUPPALA

DATE:

(321506402393)

INDEX

Contents	Page No.
Abstract	1
1. Introduction 1.1 Motivation 1.2 Problem Definition 1.3 Objective 1.4 Limitations	2 - 6
2. Survey 2.1 Introduction 2.2 Existing Systems 2.3 Disadvantages of Existing Systems 2.4 Proposed System 2.5 Advantages of Proposed System	7 - 10
3. Problem Identification and Objectives 3.1 Problem Identification 3.2 Objectives	11 -12
4. System Modelling 4.1 Introduction 4.2 Dataflow Diagram 4.3 Use Case Diagram 4.4 Class Diagram 4.5 Sequence Diagram 4.6 Activity Diagram	13 - 19
5. System Design 5.1 Introduction 5.2 System Architecture 5.3 Component Design	20 - 23
6. System Implementation 6.1 Introduction 6.2 Technology Stack 6.3 Implementation Steps	24 - 27
7. Code Implementation 7.1 Frontend 7.2 Backend 7.3 Implementation 7.4 Output	28 - 49
8. Conclusion and Future Scope	50 - 51
9. References	52

ABSTRACT

The AI Lawyer Bot and Financial Advisor is an intelligent system designed to provide basic legal support and financial advice to users. It combines the power of Artificial Intelligence (AI) and Natural Language Processing (NLP) to interact with users in a simple, conversational manner. The bot can answer queries related to laws, legal rights, and procedures, making it easier for users to understand complex legal concepts. On the financial side, the system provides guidance on budgeting, saving, investment planning, and managing personal finances. Developed to improve access to essential information, this project aims to benefit the general public, especially those who may not have easy access to professional legal or financial services.

Users can simply type their queries, and the system responds with accurate and easy-to-understand information. The goal is to reduce dependency on professionals for basic guidance and make legal and financial knowledge more accessible and affordable. The system is built using advanced AI algorithms, trained models, and a user-friendly web interface. It ensures user privacy and handles data securely. Designed for convenience, it can be accessed anytime, anywhere through a website. This tool is especially helpful for students, individuals, and small business owners in need of quick support. By combining legal and financial features, the bot acts as a digital assistant for everyday needs. The project shows how AI can promote legal awareness, improve financial literacy, and enhance user experience in a smart, scalable way.

1. INTRODUCTION

Legal and financial documents are vital components in both personal and business contexts, serving as formal records that outline agreements, obligations, and financial statuses. However, one of the significant challenges associated with these documents is their dense, technical jargon, which can be incredibly daunting for the average person to understand.

Individuals seeking legal or financial advice often find themselves faced with the labor-intensive and time-consuming task of sifting through pages of text to identify key information. This process can be frustrating and tedious, and despite their best efforts, many people struggle to fully grasp the implications of what is written. For instance, a simple misunderstanding of a clause in a contract could lead to significant legal liabilities or financial losses. Moreover, the option to consult with a legal or financial expert is not always practical for everyone. Expert advice can come at a steep cost, and many individuals may find these services prohibitively expensive, particularly if they need assistance with multiple documents.

This creates a dilemma: critical information is often buried beneath layers of technical language, and many people cannot afford the help they need to decipher it.

The need for a solution to this issue is clear. Developing a tool that can automatically extract essential data from legal and financial documents and present it in a clear, user-friendly format could transform how individuals interact with these materials. Such a tool would be especially beneficial for those who do not have a legal or financial background, allowing them to make informed decisions regarding their personal or business affairs without the constant worry of misinterpretation.

By providing easy access to relevant information, this tool could significantly reduce confusion and support users in navigating complex documents with confidence. Imagine a scenario where a small business owner could quickly understand the terms of a lease agreement or a contract without needing extensive legal training. They could focus on growing their business rather than being bogged down by the intricacies of documentation.

Additionally, such a tool could democratize access to important information, leveling the playing field for individuals who may previously have felt overwhelmed or disenfranchised by the complexities of legal and financial language. In essence, it could empower users to take control of their situations, fostering a sense of agency in their decision-making processes.

In conclusion, the challenge lies not only in the complexity of legal and financial documents but also in the accessibility of the information contained within them. The development of tools designed to simplify this complexity could be a game-changer, making essential information more available and understandable for everyone. This kind of innovation is crucial, especially as individuals increasingly seek to engage with their legal and financial realities proactively rather than reactively. By breaking down barriers to understanding, we can help ensure that everyone has the opportunity to make informed, confident decisions.

1.1 Motivation

Our motivation for this project was to simplify and improve access to legal and financial guidance for individuals who may not have easy or affordable access to professional services. Many people struggle to understand legal rights or manage their finances properly due to lack of knowledge or resources. With advancements in Artificial Intelligence, we aim to bridge this gap by creating an intelligent system that offers helpful legal and financial advice in a conversational manner.

- **Scope :**

The scope of this project is to develop an AI-powered chatbot that can understand user queries and provide accurate responses related to legal issues and financial planning. The system uses NLP techniques to process input and deliver information in a user-friendly format. It is designed to work through a web-based platform, making it accessible to a wide range of users anytime and anywhere.

- **Problem :**

The existing solutions in the market are either too complex, expensive, or limited to a single domain (either legal or financial). Many do not support real-time interaction or lack the ability to handle diverse user queries effectively. Moreover, people often rely on unverified information from the internet, which may lead to wrong decisions.

- **Solution :**

The solution is to build a smart AI chatbot capable of handling both legal and financial queries using multiple datasets and trained AI models. It provides real-time, accurate, and easy-to-understand responses to user questions. The system ensures privacy, supports secure communication, and is accessible via a web interface. This approach improves the performance, reliability, and usefulness of the system, making it a practical tool for daily use.

1.2 Problem Definition

In today's fast-paced digital world, access to reliable legal and financial advice remains a challenge for many individuals, especially those without the means to

consult professional advisors. Legal terms and financial concepts are often complex and difficult to understand, creating a knowledge gap among the general public. Moreover, existing solutions are either too expensive, domain-specific, or lack the ability to provide real-time, personalized support.

There is a growing need for an intelligent system that can offer basic legal guidance and financial planning support in a simple and accessible manner. Users should be able to get trustworthy answers to their questions without needing prior technical or legal knowledge. The current absence of such a unified and user-friendly platform creates confusion, misinformed decisions, and missed opportunities for better personal and financial well-being.

This project addresses the need by developing an AI-powered chatbot capable of understanding user queries and delivering accurate legal and financial information in real-time. The system aims to be a cost-effective, efficient, and accessible digital assistant, reducing the dependency on professionals for general advice.

1.3 OBJECTIVE

The primary objective of this project is to design and develop an AI-powered chatbot that can provide users with accurate and helpful legal support and financial guidance. The system will use Natural Language Processing (NLP) to understand user queries and respond in a conversational and easy-to-understand manner. It aims to assist users with information on basic laws, legal rights, and common legal procedures, making legal knowledge more accessible to individuals who may not have access to professional services. Additionally, the chatbot will help users manage their finances by offering advice on budgeting, saving, and investment planning, tailored to the user's needs.

The system will be accessible through a user-friendly web interface, enabling users to interact with the bot in real-time from any location. One of the key goals is to ensure that the chatbot delivers reliable and efficient responses by well-trained AI models. The project also focuses on data privacy and security, ensuring that all user interactions are handled safely. By combining legal and financial support into a single platform, the chatbot offers a cost-effective, scalable, and practical solution for everyday decision-making. This project

ultimately aims to empower users with trustworthy information and reduce their dependency on expensive or hard-to-access professional services.

1.4 LIMITATIONS

Documents format: Currently, the system supports images and PDF files. Other document formats like Word files, Excel sheets, or scanned documents with complex layouts are not supported.

AI Limitations: While OpenAI's API provides useful responses, it may not always generate perfectly accurate legal or financial advice. The system cannot replace professional consultations with legal or financial experts.

Network Access: The OpenAI API requires an internet connection, which means the application is dependent on external services. This may lead to issues if the API service is unavailable.

Scalability: As the application relies on cloud-based services like MongoDB Atlas and the OpenAI API, handling large amounts of data or numerous simultaneous users may lead to performance or cost challenges.

2. Survey

2.1 Introduction

Artificial Intelligence (AI) and Natural Language Processing (NLP) are transforming the way humans interact with machines. In recent years, these technologies have led to the development of intelligent virtual assistants and chatbots that can communicate with users, understand their queries, and provide meaningful responses. This advancement has found application in various domains such as healthcare, e-commerce, education, and banking.

In the legal field, several efforts have been made to create AI systems that can assist with legal advice. A notable example is DoNotPay, known as the world's first robot lawyer. It was initially developed to help users contest parking tickets but later expanded to cover areas like landlord disputes, immigration issues, and consumer rights. While useful, its scope is restricted to specific scenarios, and it heavily depends on predefined rules and templates rather than intelligent learning from new data.

Other research in the legal AI space explores the automation of document review, legal research, and contract analysis using machine learning. However, many of these systems are designed for legal professionals and not intended for common users seeking basic legal help.

In the financial sector, personal finance bots such as Cleo, Plum, and Digit assist users by tracking expenses, setting savings goals, and providing spending insights. These bots typically connect to users' bank accounts and use data analytics to deliver personalized financial advice. Though effective, their functionality is limited to financial planning and lacks any legal advisory capabilities.

Several academic studies highlight the potential of deep learning, recurrent neural networks (RNNs), and transformer models like BERT in improving chatbot accuracy and context understanding. Research also suggests that training bots with multiple datasets across different domains enhances their flexibility and performance.

Security and privacy are major concerns in both the legal and financial sectors. Literature emphasizes the need for secure architecture, encrypted communication, and data protection mechanisms when handling sensitive user information.

A major gap identified in the current literature is the lack of a unified platform that offers both legal and financial advice through a single intelligent system. Most existing tools focus on either legal support or financial management, which forces users to depend on multiple applications.

This project aims to bridge that gap by developing an AI-based system that integrates both legal and financial advisory services into one accessible chatbot. By utilizing NLP, AI models, and diverse datasets, the bot is designed to handle real-time queries with accuracy and ease. It builds upon the strengths of previous systems while addressing their limitations, such as limited scope, lack of real-time interaction, and insufficient integration between domains.

In conclusion, this literature review shows that while AI-powered advisory systems exist, there remains a strong need for an all-in-one intelligent assistant that can cater to both legal and financial needs in a user-friendly, secure, and scalable way.

2.2 Existing System

Manual Legal Consultation:

Individuals consult with lawyers or legal advisors for their issues. Services include personalized advice, legal document drafting, and representation in court.

Online Legal Portals:

Websites and apps provide access to legal information and connect users with lawyers. Some platforms offer automated document creation tools.

AI Technology Limitations:

AI-driven legal solutions often rely on common knowledge, which can lead to generalized or sometimes inaccurate advice.

2.3 Disadvantages of the existing system

The main disadvantage of the existing system is its low performance and efficiency. Users often find themselves spending an excessive amount of time filling out multiple forms across different websites. This repetitive process can be frustrating, leading to decreased productivity and a lack of motivation to complete applications.

Moreover, the time-consuming nature of this system can deter individuals from pursuing important opportunities. Each application may require similar information, yet users must painstakingly input data repeatedly. As a result, the inefficiency not only affects user satisfaction but can also hinder the overall progress of applications in various sectors.

2.4 Proposed System

An AI-powered legal assistant simplifies the understanding of constitutional provisions, making legal knowledge more accessible to everyone. It caters to individual needs by offering personalized legal advice for real-life problems, ensuring that users feel confident in their decisions. This innovative tool serves as a bridge between complex legal jargon and comprehensible information.

In addition to general legal guidance, the assistant provides financial legal services, including will drafting and estate management. By streamlining these processes, users can navigate important legal steps with ease. This combination of support empowers individuals to manage their legal affairs effectively and efficiently.

2.5 Advantages of Proposed System

Affordability: Reduces the cost of accessing legal advice, making it accessible to a larger audience.

Accessibility: Provides 24/7 assistance to users, including those in remote locations.

Simplicity: Breaks down complex legal language into easily understandable terms.

Comprehensive Coverage: Combines constitutional education, legal problem resolution, and financial legal advice in one platform.

Efficiency: Offers instant responses and solutions, saving users time.

Customization: Tailors advice based on the user's specific situation and needs.

3. Problem identification and Objectives

3.1 Problem Identification

Understanding legal and financial documents is a significant challenge for many individuals due to the use of technical terminology, lengthy clauses, and domain-specific jargon. These documents often require expert interpretation, which can lead to:

High consultation costs for legal and financial advice

Time-consuming manual reading and information extraction

Misunderstandings of critical terms, which can result in legal or financial consequences

Limited accessibility for individuals without a background in law or finance

Additionally, the manual process of analyzing documents in formats like images or scanned PDFs adds another layer of complexity, requiring specialized tools to extract and process the content effectively.

3.2 Objectives

Automate Text Extraction

Develop a robust system using OCR and PDF parsing to accurately extract text from a variety of document formats, including scanned images and PDFs.

Simplify Legal and Financial Language

Utilize AI-powered language models (OpenAI API) to convert complex document content into clear, human-readable summaries and explanations.

Enable Real-Time Interactions

Integrate a chatbot that can respond to user-specific questions, clarify doubts, and provide personalized assistance in real time.

Enhance User Accessibility

Create a user-friendly web interface that allows easy document upload, interaction with the chatbot, and viewing of AI-generated outputs.

Reduce Dependence on Experts

Provide an affordable and accessible alternative to expert consultations, empowering users to understand important documents on their own.

Ensure Secure Document Handling

Implement a secure backend system using ExpressJS and MongoDB to manage user data, document storage, and ensure privacy.

4. System Modelling

4.1 Introduction

Design modeling is perhaps the most crucial step in the development of the AI-Powered Legal and Finance Assistant application; this is the most important realization of the design, where a complete view can be gained on the architecture, functionalities, and workflows of the system. The end result of designing modeling techniques is to reduce the complexity of the system so that it makes sense for developers, stakeholders, and maintainers in the future. AI-Powered Legal and Finance Assistant project enjoys space to enjoy the use of Data Flow Diagrams alongside other UML diagrams, such as Use Case Diagrams, Class Diagrams, Sequence Diagrams, and even Activity Diagrams, to project and show every facet of what the system components look like-involved and who will do what during the various processes.

• Purpose:

The foremost purpose of design modeling in the AI-Powered Legal and Finance Assistant Project is to break the system down into manageable pieces: such as the front-end (React-based UI), back-end (Node.js API), and database (MongoDB). Once the components have been isolated, the focus can be put so that any component of the system responds to data flow, user interaction, or system processes with full consideration of all functional requirements from user authentication, realtime message processing, mode-based responses (legal or financial), and chat history management. The design modeling resolves conflicts in criteria and constraints, for example, in achieving the requirement of scalability, allowing multiple users, versus the need for a responsive user interface. In doing so, it provides a roadmap for the implementation phase, directing programmers in constructing a project that fulfills its aims.

• Design Goals:

The design goals for the AI-Powered Legal and Finance Assistant system target primarily the design of an application that is high quality, maintainable, and user-friendly. These goals are as follows:

- ❖ **Achieve Modularity:** Dividing the system into separate functional modules (Frontend, Backend, and Database) for simplicity and for development and enhancement in the future.
- ❖ **Ensure Scalability:** The system should be designed to ensure the carefree working of multiple users and chats at once, with all API calls and access to the database made as efficacious as possible.
- ❖ **Enhance Usability:** The User Interface should be oriented to facilitate seamless operations, including but not limited to chat history, mode selection, and file uploads.
- ❖ **Ease of Maintenance:** The system should easily be maintainable by any programmer, for instance, through the addition of new modes of response or [in-function improvement] of the functionality of the chat function within the application.

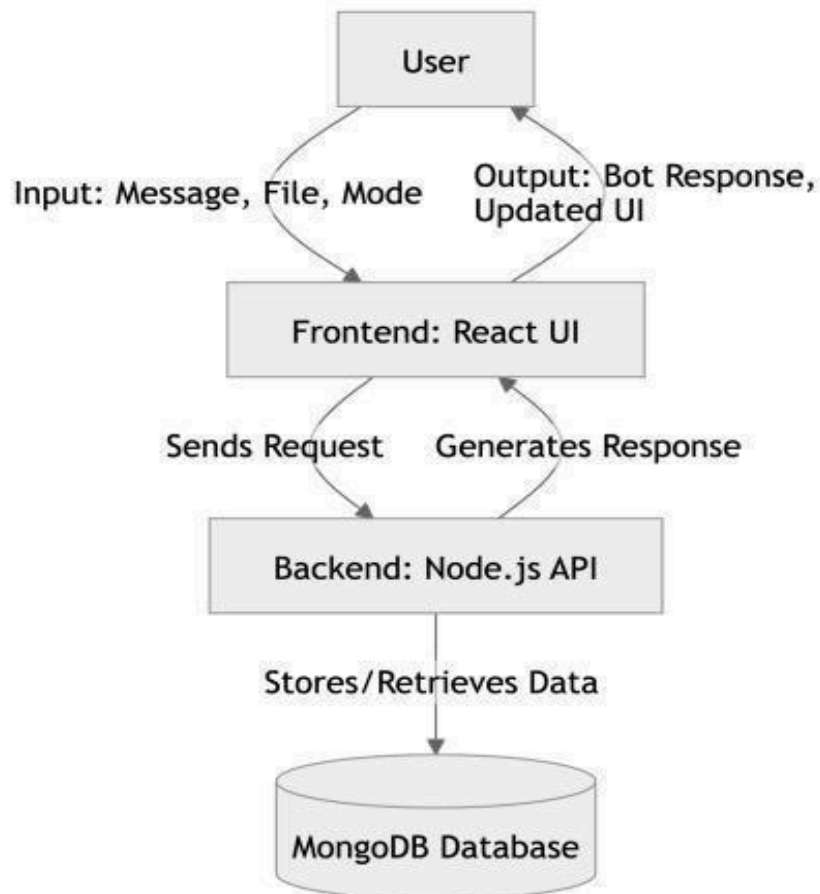
These goals assure that the AI-Powered Legal and Finance Assistant application is functional, resilient, adaptable, and maintained after deployment.

4.2 Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) also known as a bubble chart. It is a medium for inciting how data flows into and within the AI-Powered Legal and Finance Assistant system. The diagram depicts flow from the user to the system through processes and back to the user in responses. This becomes one of the crucial modeling tools that visualize system components such as the user (external entity), the frontend (React UI), the backend (Node.js API), and the database (MongoDB). It shows the input data to the system such as user messages, file uploads, mode selections, the processes from this input data, for example, message processing and generation with example outputs, for example, bot responses, updated chat histories.

The AI-Powered Legal and Finance Assistant DFD Chat is actually an end-module that shows how information flows through the system and how, at each stage, it is transformed. For instance, when a user sends a message, it is passed from the frontend to the backend processes according to the selected mode, either legal or financial. After that, it gets stored in the MongoDB

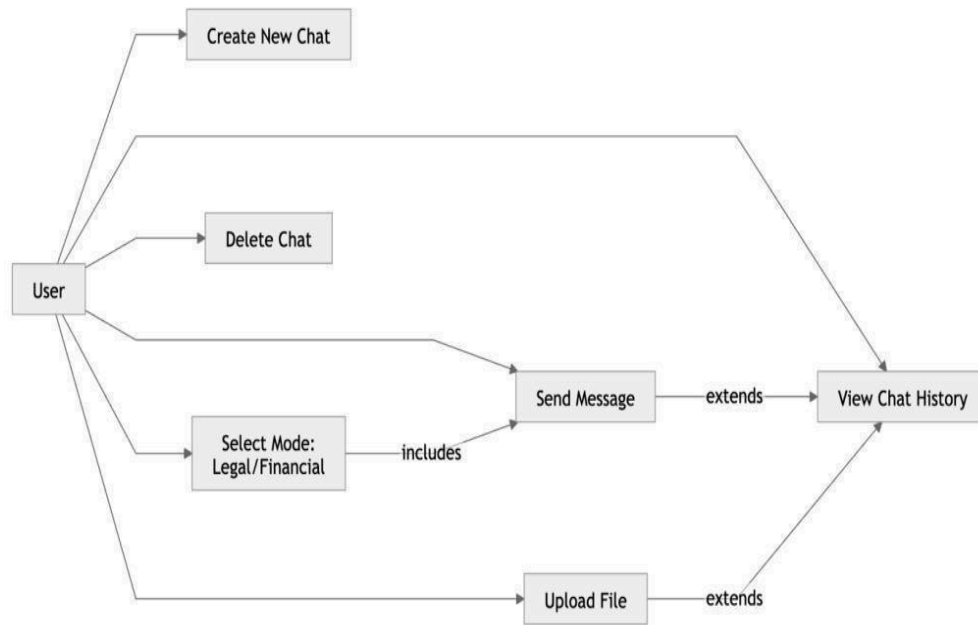
database and then responds from the frontend of the site as a bot reply. A contrived diagram can go to various levels of abstraction, but here we provide a Level-0 DFD to gain a general overview of data flow in the system. The DFD shows both developer and stakeholder user-defined behavior for functional understanding at the implementation stage, where all data interaction is considered.



4.3 Use Case Diagram

The Use Case Diagram for the AI-Powered Legal and Finance Assistant system illustrates how the "User" interacts with key functionalities. The user can create a new chat, send messages, upload files, choose a response mode (legal or financial), view chat history, and delete chats. Relationships between actions are also shown—for example, sending messages and uploading files

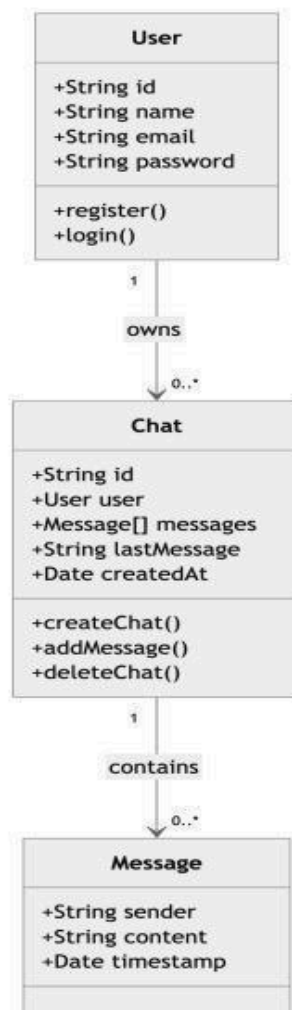
extend to viewing chat history, while selecting a mode is required before sending a message. This diagram helps define system requirements, ensuring all interactions are modeled before implementation and providing clarity for stakeholders and developers.



4.4 Class Diagram

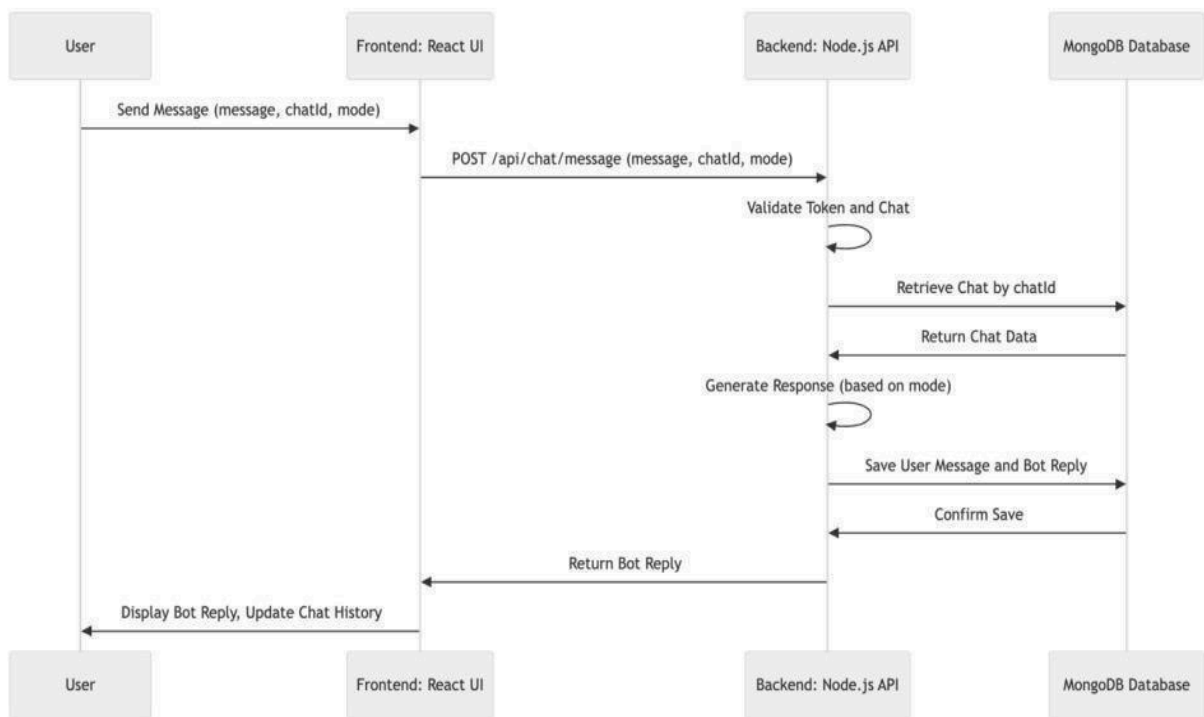
The Class Diagram is a UML diagram that represents the static structure of the AI-Powered Legal and Finance Assistant system by modeling its classes, their attributes, methods, and relationships. It provides a blueprint of the system's core components, focusing on the backend (Node.js) and database (MongoDB) models. In the AI-Powered Legal and Finance Assistant project, the key classes include User, Chat, and Message.

The User class contains attributes like id, name, email, and password, along with methods like register() and login(). The Chat class includes attributes such as id, user, messages, lastMessage, and createdAt, with methods like createChat(), addMessage(), and deleteChat(). The Message class has attributes like sender, content, and timestamp.



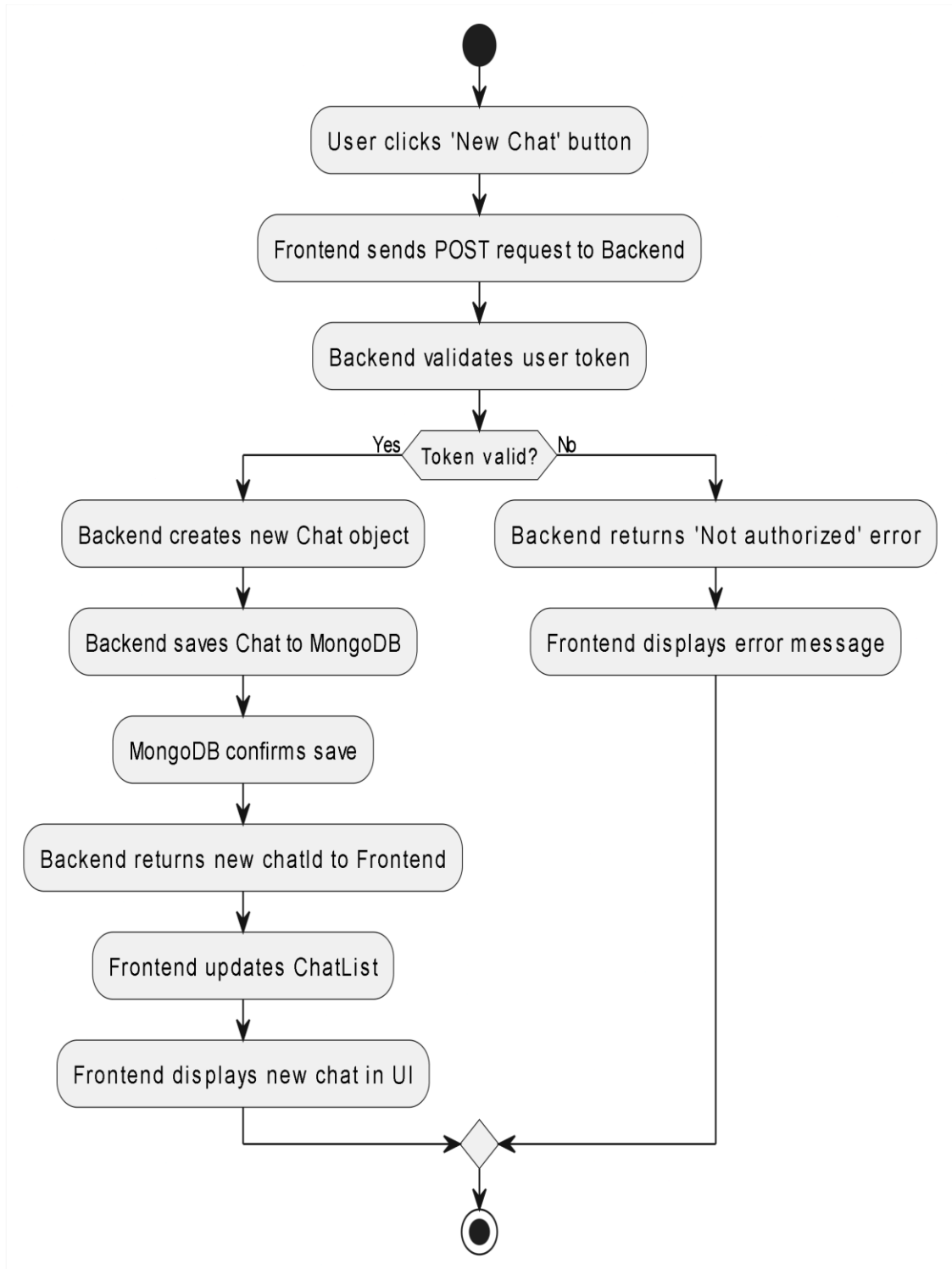
4.5 Sequence Diagram

The Sequence Diagram is a UML diagram demonstrating the dynamic behavior of the AI-Powered Legal and Finance Assistant system by showing the sequence of interactions between different components during a certain flow; in this case, sending a message. It shows how the user, frontend (React UI), backend (Node.js API), and database (MongoDB) interact with each other over time to process a user's message and deliver the answer. This diagram shows the message flow (e.g., API requests and responses) between the components as well as the order of these interactions.



4.6 Activity Diagram

The Activity Diagram is a UML diagram that models the workflow of a specific process within the AI-Powered Legal and Finance Assistant system in this case, the process of creating a new chat. It provides a step by step representation of the activities involved, including decision points, actions, and transitions between states. The diagram is particularly useful for visualizing the system's procedural logic and ensuring that all steps in the process are accounted for.



5. System Design

5.1 Introduction

In this section, we present the system design of our AI-Powered Legal and Finance Assistant project, which serves as the foundation for implementing our application. System design is a crucial step in our development process, as it outlines how the various components of the system interact to achieve the desired functionality. Our team has focused on designing a system that supports all the features of the AI-Powered Legal and Finance Assistant application, such as creating chats, processing user messages, generating responses using the Open AI API, and managing chat history. By carefully designing the system, we aim to ensure that it meets the functional requirements while being efficient and maintainable.

Purpose:

The purpose of the system design is to provide a detailed plan for the implementation of the AI-Powered Legal and Finance Assistant application. This involves defining the architecture, components, and interactions between the frontend, backend, database, and external APIs like the Open AI API, which we use to generate intelligent responses based on the user's selected mode (legal or financial). The design process helps us address challenges such as user authentication, real-time message processing, and secure data storage. By creating a wellstructured system design, we ensure that the application is robust, scalable, and capable of delivering a seamless user experience.

Design Goals:

Our team has established several design goals to guide the development of the AI-Powered Legal and Finance Assistant system: Modularity: Design the system in a modular way, separating the frontend, backend, and database components to simplify development and future enhancements.

Efficiency:

Ensure that the system processes user requests, such as message generation via the Open AI API, quickly and efficiently to provide a responsive user

experience. **Security:** Implement user authentication and authorization to protect user data and ensure that only authorized users can access their chats. **Maintainability:** Create a system that is easy to maintain and update, allowing us to add new features, such as additional response modes or improved API integrations, with minimal effort. These goals have been instrumental in shaping our system design and ensuring that the AI-Powered Legal and Finance Assistant application meets the project requirements.

5.2 System Architecture

The AI-Powered Legal and Finance Assistant system is designed using a client-server architecture, which consists of three main layers: the frontend, the backend, and the database, along with an external integration with the Open AI API for response generation. This architecture ensures that each layer has a specific role, allowing for efficient communication and data flow between components. Below, we describe the key components of the system architecture:

- **Frontend (React UI):** The frontend is built using React, a JavaScript library for creating dynamic user interfaces. It serves as the client-side layer where users interact with the application. The frontend includes components such as the chat window, sidebar for chat history, and mode selection interface (legal or financial). It communicates with the backend by sending HTTP requests, such as creating a new chat or sending a message.

- **Backend (Node.js API):** The backend is developed using Node.js with the Express framework, acting as the server-side layer that handles all business logic. It processes requests from the frontend, such as user authentication, chat creation, and message processing. The backend also integrates with the Open AI API to generate responses based on the user's message and selected mode. For instance, if a user selects "legal" mode and sends a message, the backend sends the message to the Open AI API, retrieves a legal-focused response, and returns it to the frontend.

- **Database (MongoDB):** MongoDB, a NoSQL database, is used to store persistent data, including user information, chats, and messages. The backend interacts with MongoDB to save new messages, retrieve chat history, and update chat details.

- **Open AI API:** The Open AI API is an external service integrated into the backend to provide intelligent response generation. The backend sends the user's message and mode (legal or financial) to the Open AI API, which processes the input and returns a contextually appropriate response. This integration enhances the functionality of the AI-Powered Legal and Finance Assistant by enabling it to provide meaningful and mode-specific replies to user queries. The system architecture ensures a clear separation of concerns, with the frontend handling user interactions, the backend managing logic and API integrations, and the database storing data. The Open AI API adds an intelligent layer to the system, making the AI-Powered Legal and Finance Assistant capable of delivering advanced responses.

5.3 Component Design

In this subsection, we describe the design of the main components of the AI-Powered Legal and Finance Assistant system, focusing on their roles and interactions. The system is divided into modular components to ensure that each part has a specific responsibility, making the application easier to develop and maintain. Below, we outline the key components:

Frontend Components:

- **ChatList Component:** This component displays a list of the user's chats in the sidebar, showing the last message of each chat for quick reference. Users can select a chat to view its history or create a new chat using a dedicated button.
- **ChatWindow Component:** The ChatWindow component is the main interface for user-bot interaction. It allows users to type messages, select a mode (legal or financial), and upload files. It also displays the conversation history, including messages from both the user and the bot.
- **ModeSelector Component:** This component provides an interface for users to choose between legal and financial modes. The selected mode is sent to the backend along with the user's message to ensure that the Open AI API generates a response in the appropriate context.

Backend Components:

- **User Routes:** These routes handle user-related operations, such as registration (POST /api/users/register) and login (POST /api/users/login). JSON Web Tokens (JWT) are used for authentication to ensure that only authorized users can access their chats.

- **Chat Routes:** These routes manage chat-related operations, including creating a new chat (POST /api/chat/new), sending a message (POST /api/chat/message), and deleting a chat (DELETE /api/chat/:id). When a message is sent, the backend processes the request by calling the Open AI API to generate a response, saves the message and response to the database, and returns the response to the frontend.

- **Open AI API Integration:** The backend includes a module to interact with the Open AI API. This module sends the user's message and mode to the API, retrieves the generated response, and handles any errors that may occur during the API call. This integration is critical for providing intelligent and context-aware responses to user queries.

Database Models:

- **User Model:** Stores user data, including id, name, email, and password.

- **Chat Model:** Stores chat data, including id, user (reference to the user who owns the chat), messages (an array of messages), lastMessage, and createdAt.

- **Message Model:** Embedded within the Chat model, this stores message details such as sender (user or bot), content, and timestamp.

These components are designed to work together efficiently, ensuring that user requests are processed seamlessly from the frontend to the backend, with the Open AI API enhancing the system's response capabilities.

6. System Implementation

6.1 Introduction

This section outlines the implementation phase of our Law Bot project, where we translated the system design into a fully functional application. The implementation process involved building the frontend, backend, and database components, as well as integrating the Open AI API to enable intelligent response generation. Our team focused on ensuring that the system meets all functional requirements, such as user authentication, chat creation, message processing, and mode-based responses (legal or financial). This section provides a detailed overview of the technologies used, the implementation steps, and the challenges we encountered during development.

Purpose:

The purpose of this section is to document the practical steps taken to implement the Law Bot system. By detailing the implementation process, we aim to provide a clear understanding of how the system was built, from setting up the development environment to coding the core functionalities. This includes the integration of the Open AI API, which plays a critical role in generating context-aware responses based on the user's selected mode. The implementation phase also allowed us to test our design decisions and address any issues that arose, ensuring that the final system aligns with the project objectives.

Implementation Goals: Our team established the following goals for the implementation phase:

- **Functionality:** Ensure that all features, such as chat creation, message processing, and mode selection, are fully implemented and operational.
- **Integration:** Successfully integrate the Open AI API to provide intelligent responses, enhancing the user experience.
- **Reliability:** Build a system that operates reliably, with proper error handling for API calls, database operations, and user interactions.
- **Documentation:** Maintain clear documentation of the code and implementation process to facilitate future maintenance and updates.

These goals guided our efforts and helped us deliver a working application that meets the project requirements.

6.2 Technology Stack

To implement the Law Bot system, our team selected a technology stack that aligns with the system design and supports the project's requirements. Below, we describe the key technologies used for each component of the system:

Frontend:

- **React:** We used React, a JavaScript library, to build the frontend user interface. React's component-based architecture allowed us to create reusable components like the chat window and sidebar, making the development process more efficient.
- **Axios:** Axios was used to make HTTP requests from the frontend to the backend API, such as sending messages or creating new chats.
- **React Router:** React Router was employed for navigation between pages, such as the login page and the main chat interface.

Backend:

- **Node.js and Express:** The backend was developed using Node.js with the Express framework, which provided a robust foundation for building RESTful APIs. Express handled routing and middleware for user authentication and request processing.
- **Mongoose:** Mongoose, an Object Data Modeling (ODM) library, was used to interact with MongoDB, simplifying database operations like creating, reading, updating, and deleting data.
- **JSON Web Tokens (JWT):** JWT was implemented for user authentication, ensuring that only authorized users can access their chats.
- **Axios:** Axios was also used in the backend to make HTTP requests to the Open AI API for generating responses.

Database:

- **MongoDB:** MongoDB, a NoSQL database, was chosen for its flexibility in handling unstructured data like chats and messages. It stores user data, chat details, and conversation history.

External API:

- **Open AI API:** The Open AI API was integrated into the backend to generate intelligent responses based on the user's message and selected mode (legal or financial). This API enhances the system's ability to provide contextually relevant replies, making the Law Bot more interactive and useful.

This technology stack was selected based on its compatibility with our system design and the team's familiarity with these tools, ensuring a smooth implementation process.

6.3 Implementation Steps

The implementation of the Law Bot system was carried out in several steps, each focusing on a specific component or functionality. Below, we describe the key steps involved:

- **Step 1 - Setting Up the Development Environment:** We began by setting up the development environment for both the frontend and backend. For the frontend, we initialized a React project using create-react-app, installed dependencies like Axios and React Router, and set up the project structure with components like ChatList, ChatWindow, and ModeSelector. For the backend, we created a Node.js project, installed Express, Mongoose, and other dependencies, and configured the server to connect to MongoDB. We also set up environment variables to securely store sensitive information, such as the Open AI API key and MongoDB connection string.

- **Step 2 - Implementing User Authentication:** User authentication was implemented using JWT. We created routes for user registration (POST /api/users/register) and login (POST /api/users/login), which generate a JWT token upon successful authentication. This token is sent to the frontend and included in the header of subsequent requests to protected routes, such as

creating a chat or sending a message. A middleware function was added to verify the token and ensure that only authorized users can access their data.

- **Step 3 - Building the Database Models Using Mongoose:** we defined schemas for the User and Chat models. The User model includes fields like id, name, email, and password, while the Chat model includes id, user (a reference to the user), messages (an array of embedded message objects), lastMessage, and createdAt. The Message schema, embedded within Chat, includes fields like sender, content, and timestamp. These models were used to perform CRUD operations, such as saving new messages or retrieving chat history.

- **Step 4 - Implementing the Backend API:** The backend API was developed using Express, with routes for user and chat operations.

Key routes include:

- **POST /api/chat/new:** Creates a new chat for the authenticated user.
- **POST /api/chat/message:** Processes a user's message, calls the Open AI API to generate a response, saves the message and response to the database, and returns the response to the frontend.
- **DELETE /api/chat/:id:** Deletes a chat for the authenticated user.

- **Step 5 - Developing the Frontend Interface:** The frontend was built using React, with components designed to provide a user-friendly interface. The ChatList component displays a list of chats, the ChatWindow component handles message input and display, and the ModeSelector component allows users to choose between legal and financial modes. Axios was used to make API calls to the backend, such as sending a message or creating a new chat. The frontend also updates dynamically to reflect changes, such as displaying new messages or updating the chat list.

- **Step 6 - Testing and Debugging:** Throughout the implementation, we tested each component to ensure it worked as expected. We used tools like Postman to test the backend API endpoints and the browser's developer tools to debug the frontend. We also tested the Open AI API integration by sending sample messages in different modes and verifying the responses. Any issues, such as failed API calls or database errors, were addressed during this phase.

7. Code Implementation

7.1 Frontend

App.js

```
import React, { useEffect } from 'react';
import { BrowserRouter as Router, Route, Routes, Navigate, useLocation } from
'react-router-dom';
import Home from './pages/Home';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import ChatPage from './pages/ChatPage'; // Should point to the new ChatPage.js
import Navbar from './components/Navbar';
import { ThemeProvider, useTheme } from './context/ThemeContext';
import AdminPage from './pages/AdminPage';
import AdminLogin from './pages/AdminLogin';
import './App.css';
const AppContent = () => {
  const { isDarkMode } = useTheme();
  const location = useLocation();
  const isAdminRoute = location.pathname.startsWith('/admin');

  useEffect(() => {
    document.documentElement.setAttribute('data-theme', isDarkMode ? 'dark' :
'light');
  }, [isDarkMode]);

  return (
    <>
      {!isAdminRoute && <Navbar />}
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<LoginPage />} />
        <Route path="/register" element={<RegisterPage />} />
        <Route path="/chat" element={<ChatPage />} />
        <Route path="/admin/login" element={<AdminLogin />} />
        <Route
          path="/admin"
          element={
            localStorage.getItem('adminToken') ? (
              <AdminPage />
            ) : (
              <Navigate to="/admin/login" />
            )
          }
        />
      </Routes>
    </>
  );
};

const App = () => {
  return (
    <ThemeProvider><Router>
      <AppContent />
    </Router>
  </ThemeProvider>
  );
};
export default App;
```


AdminLogin.js

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import {
  Container,
  Paper,
  TextField,
  Button,
  Typography,
  Box,
  CircularProgress,
} from '@mui/material';

const AdminLogin = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  useEffect(() => {
    const checkAuth = async () => {
      const token = localStorage.getItem('adminToken');
      if (token) {
        try {
          await axios.get('http://localhost:4500/api/admin/verify', {
            headers: { 'x-auth-token': token }
          });
          navigate('/admin');
        } catch (error) {
          localStorage.removeItem('adminToken');
        }
      }
    };
    checkAuth();
  }, [navigate]);

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError('');

    try {
      const response = await axios.post('http://localhost:4500/api/admin/login',
formData);
      localStorage.setItem('adminToken', response.data.token);
      navigate('/admin');
    } catch (error) {
      setError(error.response?.data?.message || 'Invalid credentials');
    } finally {
      setLoading(false);
    }
  };

  if (loading) {
```

```

    return (
      <Box display="flex" justifyContent="center" alignItems="center"
minHeight="100vh">
        <CircularProgress />
      </Box>
    );
  }

  return (
    <Container maxWidth="sm">
      <Box sx={{ mt: 8 }}>
        <Paper elevation={3} sx={{ p: 4 }}>
          <Typography variant="h4" align="center" gutterBottom>
            Admin Login
          </Typography>
          {error && (
            <Typography color="error" align="center" gutterBottom>
              {error}
            </Typography>
          )}
          <form onSubmit={handleSubmit}>
            <TextField
              fullWidth
              label="Email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              margin="normal"
              required
            />
            <TextField
              fullWidth
              label="Password"
              name="password"
              type="password"
              value={formData.password}
              onChange={handleChange}
              margin="normal"
              required
            />
            <Button
              type="submit"
              fullWidth
              variant="contained"
              color="primary"
              sx={{ mt: 3 }}
              disabled={loading}
            >
              Login
            </Button>
          </form>
        </Paper>
      </Box>
    </Container>
  );
};

export default AdminLogin;

```

AdminPage.js

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import {
  Container,
  Paper,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  Button,
  Dialog,
  DialogTitle,
  DialogContent,
  DialogActions,
  TextField,
  Typography,
  Box,
  IconButton,
  SnackBar,
  Alert,
  CircularProgress,
  useTheme,
} from '@mui/material';
import AddIcon from '@mui/icons-material/Add';
import EditIcon from '@mui/icons-material/Edit';
import DeleteIcon from '@mui/icons-material/Delete';
import HistoryIcon from '@mui/icons-material/History';
import Brightness4Icon from '@mui/icons-material/Brightness4';
import Brightness7Icon from '@mui/icons-material/Brightness7';
import { useTheme as useCustomTheme } from '../context/ThemeContext';
import '../styles/AdminPage.css';

const AdminNavbar = ({ isDarkMode, toggleTheme, onAddUser }) => {
  return (
    <nav className="admin-navbar">
      <div className="admin-nav-left">
        <h1 className="admin-nav-title">Admin Dashboard</h1>
      </div>
      <div className="admin-nav-right">
        <IconButton onClick={toggleTheme} color="inherit">
          {isDarkMode ? <Brightness7Icon /> : <Brightness4Icon />}
        </IconButton>
        <Button
          variant="contained"
          color="primary"
          startIcon={<AddIcon />}
          onClick={onAddUser}
          sx={{
            backgroundColor: 'primary.main',
            color: 'primary.contrastText',
            '&:hover': {
              backgroundColor: 'primary.dark',
            }
          }}
        >
          Add User
        </Button>
      </div>
    </nav>
  );
};
```

```

const AdminPage = () => {
  const theme = useTheme();
  const { isDarkMode, toggleTheme } = useCustomTheme();
  const [users, setUsers] = useState([]);
  const [selectedUser, setSelectedUser] = useState(null);
  const [searchHistory, setSearchHistory] = useState([]);
  const [openEditDialog, setOpenEditDialog] = useState(false);
  const [openAddDialog, setOpenAddDialog] = useState(false);
  const [openHistoryDialog, setOpenHistoryDialog] = useState(false);
  const [editForm, setEditForm] = useState({ name: "", email: "", password: "" });
  const [addForm, setAddForm] = useState({ name: "", email: "", password: "" });
  const [snackbar, setSnackbar] = useState({ open: false, message: "", severity: 'success' });
  const [loading, setLoading] = useState(true);
  const [deleteDialog, setDeleteDialog] = useState({
    open: false,
    userId: null,
    userName: ""
  });
  const navigate = useNavigate();

  useEffect(() => {
    const checkAuth = async () => {
      const token = localStorage.getItem('adminToken');
      if (!token) {
        navigate('/admin/login');
        return;
      }

      try {
        // Verify token with backend
        await axios.get('http://localhost:4500/api/admin/verify', {
          headers: { 'x-auth-token': token }
        });
        await fetchUsers();
      } catch (error) {
        localStorage.removeItem('adminToken');
        navigate('/admin/login');
      } finally {
        setLoading(false);
      }
    };

    checkAuth();
  }, [navigate]);

  const fetchUsers = async () => {
    try {
      const token = localStorage.getItem('adminToken');
      const response = await axios.get('http://localhost:4500/api/admin/users', {
        headers: { 'x-auth-token': token }
      });
      setUsers(response.data);
    } catch (error) {
      showSnackbar('Error fetching users', 'error');
    }
  };

  const handleAdd = () => {
    setAddForm({ name: "", email: "", password: "" });
    setOpenAddDialog(true);
  };

  const handleAddSubmit = async () => {
    try {
      const token = localStorage.getItem('adminToken');

```

```

    await axios.post('http://localhost:4500/api/admin/users', addForm, {
      headers: { 'x-auth-token': token }
    });
    fetchUsers();
    setOpenAddDialog(false);
    showSnackbar('User added successfully', 'success');
  } catch (error) {
    showSnackbar('Error adding user', 'error');
  }
};

const handleEdit = (user) => {
  setSelectedUser(user);
  setEditForm({ name: user.name, email: user.email, password: '' });
  setOpenEditDialog(true);
};

const handleUpdate = async () => {
  try {
    const token = localStorage.getItem('adminToken');
    await axios.put('http://localhost:4500/api/admin/users/${selectedUser._id}', editForm, {
      headers: { 'x-auth-token': token }
    });
    fetchUsers();
    setOpenEditDialog(false);
    showSnackbar('User updated successfully', 'success');
  } catch (error) {
    showSnackbar('Error updating user', 'error');
  }
};

const handleDelete = (userId, userName) => {
  setDeleteDialog({
    open: true,
    userId,
    userName
  });
};

const handleDeleteConfirm = async () => {
  try {
    const token = localStorage.getItem('adminToken');
    await axios.delete('http://localhost:4500/api/admin/users/${deleteDialog.userId}', {
      headers: { 'x-auth-token': token }
    });
    fetchUsers();
    showSnackbar('User deleted successfully', 'success');
  } catch (error) {
    console.error('Error deleting user:', error);
    showSnackbar(error.response?.data?.message || 'Error deleting user', 'error');
  } finally {
    setDeleteDialog({ open: false, userId: null, userName: '' });
  }
};

const handleDeleteCancel = () => {
  setDeleteDialog({ open: false, userId: null, userName: '' });
};

const handleViewHistory = async (userId) => {
  try {
    const token = localStorage.getItem('adminToken');
    console.log('Fetching search history for user:', userId);

    const response = await axios.get('http://localhost:4500/api/admin/users/${userId}/search-history', {
      headers: { 'x-auth-token': token }
    });
  }
};

```

```

});

console.log('Search history response:', response.data);

if (response.data && response.data.length > 0) {
  setSearchHistory(response.data);
  setOpenHistoryDialog(true);
} else {
  showSnackbar('No search history found for this user', 'info');
}
} catch (error) {
  console.error('Error fetching search history:', error);
  showSnackbar(error.response?.data?.message || 'Error fetching search history', 'error');
}
};

const showSnackbar = (message, severity) => {
  setSnackbar({ open: true, message, severity });
};

if (loading) {
  return (
    <Box display="flex" justifyContent="center" alignItems="center" minHeight="100vh">
      <CircularProgress />
    </Box>
  );
}

return (
  <div className="admin-page-container">
    <AdminNavbar
      isDarkMode={isDarkMode}
      toggleTheme={toggleTheme}
      onAddUser={handleAdd}
    />
    <div className="admin-content">
      <TableContainer
        component={Paper}
        className="admin-table-container"
        sx={{
          backgroundColor: 'background.paper',
          boxShadow: 'var(--card-shadow)',
          border: '1px solid var(--link-color)',
          borderRadius: 'var(--border-radius)',
          maxHeight: 'calc(100vh - 200px)',
          overflow: 'auto'
        }}
      >
        <Table stickyHeader>
          <TableHead>
            <TableRow>
              <TableCell sx={{
                color: 'text.primary',
                fontWeight: 'bold',
                backgroundColor: 'background.paper',
                borderBottom: '2px solid var(--link-color)'
              }}>Name</TableCell>
              <TableCell sx={{
                color: 'text.primary',
                fontWeight: 'bold',
                backgroundColor: 'background.paper',
                borderBottom: '2px solid var(--link-color)'
              }}>Email</TableCell>
              <TableCell sx={{
                color: 'text.primary',
                fontWeight: 'bold',

```

```

        backgroundColor: 'background.paper',
        borderBottom: '2px solid var(--link-color)'
    }}>Actions</TableCell>
</TableRow>
</TableHead>
<TableBody>
  {users.map((user) => (
    <TableRow key={user._id} sx={{
      '&:hover': {
        backgroundColor: 'action.hover'
      }
    }}>
      <TableCell sx={{
        color: 'text.primary',
        borderBottom: '1px solid var(--link-color)'
      }}>{user.name}</TableCell>
      <TableCell sx={{
        color: 'text.primary',
        borderBottom: '1px solid var(--link-color)'
      }}>{user.email}</TableCell>
      <TableCell sx={{
        borderBottom: '1px solid var(--link-color)'
      }}>
        <IconButton
          onClick={() => handleEdit(user)}
          sx={{
            color: 'text.primary',
            '&:hover': {
              color: 'text.secondary'
            }
          }}
        >
        <EditIcon />
      </IconButton>
      <IconButton
        onClick={() => handleDelete(user._id, user.name)}
        sx={{
          color: 'error.main',
          '&:hover': {
            color: 'error.dark'
          }
        }}
      >
      <DeleteIcon />
    </IconButton>
    <IconButton
      onClick={() => handleViewHistory(user._id)}
      sx={{
        color: 'info.main',
        '&:hover': {
          color: 'info.dark'
        }
      }}
    >
    <HistoryIcon />
  </IconButton>
</TableCell>
</TableRow>
  ))}
</TableBody>
</Table>
</TableContainer>

{ /* Delete Confirmation Dialog */ }
<Dialog
  open={deleteDialog.open}

```

```

onClose={handleDeleteCancel}
PaperProps={{
  sx: {
    backgroundColor: 'background.paper',
    color: 'text.primary',
    borderRadius: 'var(--border-radius)',
    minWidth: '400px',
    maxWidth: '500px'
  }
}}
>
<DialogTitle sx={{
  color: 'text.primary',
  borderBottom: '1px solid var(--link-color)',
  backgroundColor: 'background.paper'
}}>
  Confirm Delete
</DialogTitle>
<DialogContent sx={{
  backgroundColor: 'background.paper',
  padding: '20px'
}}>
  <Typography sx={{ color: 'text.primary' }}>
    Are you sure you want to delete user: {deleteDialog.userName}?
  </Typography>
</DialogContent>
<DialogActions sx={{
  backgroundColor: 'background.paper',
  borderTop: '1px solid var(--link-color)',
  padding: '16px'
}}>
  <Button
    onClick={handleDeleteCancel}
    sx={{ color: 'text.primary' }}
  >
    Cancel
  </Button>
  <Button
    onClick={handleDeleteConfirm}
    color="error"
    sx={{ color: 'error.main' }}
  >
    Delete
  </Button>
</DialogActions>
</Dialog>

{/* Add User Dialog */}
<Dialog
  open={openAddDialog}
  onClose={() => setOpenAddDialog(false)}
  PaperProps={{
    sx: {
      backgroundColor: 'background.paper',
      color: 'text.primary',
      borderRadius: 'var(--border-radius)'
    }
  }}
>
  <DialogTitle sx={{ color: 'text.primary' }}>Add New User</DialogTitle>
  <DialogContent>
    <TextField
      margin="dense"
      label="Name"
      fullWidth
      value={addForm.name}
    >

```



```

    onChange={(e) => setAddForm({ ...addForm, name: e.target.value })}
    sx={{
      '& .MuiInputLabel-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
    }}
  />
  <TextField
    margin="dense"
    label="Email"
    fullWidth
    value={addForm.email}
    onChange={(e) => setAddForm({ ...addForm, email: e.target.value })}
    sx={{
      '& .MuiInputLabel-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
    }}
  />
  <TextField
    margin="dense"
    label="Password"
    type="password"
    fullWidth
    value={addForm.password}
    onChange={(e) => setAddForm({ ...addForm, password: e.target.value })}
    sx={{
      '& .MuiInputLabel-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-root': { color: 'text.primary' },
      '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
    }}
  />
</DialogContent>
<DialogActions>
  <Button
    onClick={() => setOpenAddDialog(false)}
    sx={{ color: 'text.primary' }}
  >
    Cancel
  </Button>
  <Button
    onClick={handleAddSubmit}
    variant="contained"
    sx={{
      backgroundColor: 'primary.main',
      color: 'primary.contrastText',
      '&:hover': {
        backgroundColor: 'primary.dark',
      }
    }}
  >
    Add
  </Button>
</DialogActions>
</Dialog>

{/* Edit User Dialog */}
<Dialog
  open={openEditDialog}
  onClose={() => setOpenEditDialog(false)}
  PaperProps={{
    sx: {
      backgroundColor: 'background.paper',
      color: 'text.primary',
      borderRadius: 'var(--border-radius)'
    }
  }}
  />

```

```

    }}
  >
  <DialogTitle sx={{ color: 'text.primary' }}>Edit User</DialogTitle>
  <DialogContent>
    <TextField
      margin="dense"
      label="Name"
      fullWidth
      value={editForm.name}
      onChange={(e) => setEditForm({ ...editForm, name: e.target.value })}
      sx={{
        '& .MuiInputLabel-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
      }}
    />
    <TextField
      margin="dense"
      label="Email"
      fullWidth
      value={editForm.email}
      onChange={(e) => setEditForm({ ...editForm, email: e.target.value })}
      sx={{
        '& .MuiInputLabel-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
      }}
    />
    <TextField
      margin="dense"
      label="New Password (leave blank to keep current)"
      type="password"
      fullWidth
      value={editForm.password}
      onChange={(e) => setEditForm({ ...editForm, password: e.target.value })}
      sx={{
        '& .MuiInputLabel-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-root': { color: 'text.primary' },
        '& .MuiOutlinedInput-notchedOutline': { borderColor: 'text.primary' }
      }}
    />
  </DialogContent>
  <DialogActions>
    <Button
      onClick={() => setOpenEditDialog(false)}
      sx={{ color: 'text.primary' }}
    >
      Cancel
    </Button>
    <Button
      onClick={handleUpdate}
      variant="contained"
      sx={{
        backgroundColor: 'primary.main',
        color: 'primary.contrastText',
        '&:hover': {
          backgroundColor: 'primary.dark',
        }
      }}
    >
      Update
    </Button>
  </DialogActions>
</Dialog>

{/* Search History Dialog */}

```

```

<Dialog
  open={openHistoryDialog}
  onClose={() => setOpenHistoryDialog(false)}
  maxWidth="md"
  fullWidth
  PaperProps={{
    sx: {
      backgroundColor: isDarkMode ? '#000000' : 'background.paper',
      color: isDarkMode ? '#ffffff' : 'text.primary',
      borderRadius: 'var(--border-radius)',
      maxHeight: '80vh'
    }
  }}
>
  <DialogTitle sx={{
    color: isDarkMode ? '#ffffff' : 'text.primary',
    borderBottom: '1px solid var(--link-color)',
    backgroundColor: isDarkMode ? '#000000' : 'background.paper'
  }}>
    Search History
  </DialogTitle>
  <DialogContent sx={{
    backgroundColor: isDarkMode ? '#000000' : 'background.paper',
    padding: '20px'
  }}>
    {searchHistory.length === 0 ? (
      <Typography align="center" sx={{ py: 4, color: isDarkMode ? '#ffffff' : 'text.primary' }}>
        No search history available
      </Typography>
    ) : (
      <TableContainer sx={{
        backgroundColor: isDarkMode ? '#000000' : 'background.paper',
        maxHeight: '60vh'
      }}>
        <Table stickyHeader>
          <TableHead>
            <TableRow>
              <TableCell sx={{
                color: isDarkMode ? '#ffffff' : 'text.primary',
                fontWeight: 'bold',
                backgroundColor: isDarkMode ? '#000000' : 'background.paper',
                borderBottom: '2px solid var(--link-color)'
              }}>Search Query</TableCell>
              <TableCell sx={{
                color: isDarkMode ? '#ffffff' : 'text.primary',
                fontWeight: 'bold',
                backgroundColor: isDarkMode ? '#000000' : 'background.paper',
                borderBottom: '2px solid var(--link-color)'
              }}>Timestamp</TableCell>
              <TableCell sx={{
                color: isDarkMode ? '#ffffff' : 'text.primary',
                fontWeight: 'bold',
                backgroundColor: isDarkMode ? '#000000' : 'background.paper',
                borderBottom: '2px solid var(--link-color)'
              }}>Results</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {searchHistory.map((history) => (
              <TableRow key={history._id} sx={{
                '&:hover': {
                  backgroundColor: isDarkMode ? '#1a1a1a' : 'action.hover'
                }
              }}>
                <TableCell sx={{
                  color: isDarkMode ? '#ffffff' : '#000000',

```

```

borderBottom: '1px solid var(--link-color)'
}}>{history.searchQuery}</TableCell>
<TableCell sx={{
  color: isDarkMode ? '#ffffff' : '#000000',
  borderBottom: '1px solid var(--link-color)'
}}>{new Date(history.timestamp).toLocaleString()}</TableCell>
<TableCell sx={{
  color: isDarkMode ? '#ffffff' : '#000000',
  borderBottom: '1px solid var(--link-color)'
}}>
{history.results && history.results.length > 0 ? (
  history.results.map((result, index) => (
    <div key={index} style={{
      marginBottom: '8px',
      whiteSpace: 'pre-wrap',
      color: isDarkMode ? '#ffffff' : '#000000'
    }}>
      {result}
    </div>
  ))
) : (
  <Typography sx={{ color: isDarkMode ? '#ffffff' : '#000000' }}>No results</Typography>
)}
</TableCell>
</TableRow>
))}
</TableBody>
</Table>
</TableContainer>
)}
</DialogContent>
<DialogActions sx={{
  backgroundColor: isDarkMode ? '#000000' : 'background.paper',
  borderTop: '1px solid var(--link-color)',
  padding: '16px'
}}>
  <Button
    onClick={() => setOpenHistoryDialog(false)}
    sx={{ color: isDarkMode ? '#ffffff' : 'text.primary' }}
  >
    Close
  </Button>
</DialogActions>
</Dialog>

{/* Snackbar for notifications */}
<Snackbar
  open={snackbar.open}
  autoHideDuration={6000}
  onClose={() => setSnackbar({ ...snackbar, open: false })}
>
  <Alert
    onClose={() => setSnackbar({ ...snackbar, open: false })}
    severity={snackbar.severity}
    sx={{ width: '100%' }}
  >
    {snackbar.message}
  </Alert>
</Snackbar>
</div>
</div>
);
};

```

export default AdminPage;

Login.js

```
import React, { useState, useContext } from 'react';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import '../styles/Login.css';

const Login = () => {
  const [formData, setFormData] = useState({ email: '', password: '' });
  const [error, setError] = useState('');
  const { login } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post('http://localhost:4500/api/auth/login', formData);
      login(res.data.token);
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.msg || 'Login failed');
    }
  };

  return (
    <div className="login">
      <h2>Login</h2>
      {error && <p className="error">{error}</p>}
      <form onSubmit={handleSubmit}>
        <div>
          <label>Email:</label>
          <input type="email" name="email" value={formData.email} onChange={handleChange} required />
        </div>
        <div>
          <label>Password:</label>
          <input type="password" name="password" value={formData.password} onChange={handleChange}
required />
        </div>
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default Login;
```

Register.js

```
import React, { useState, useContext } from 'react';
import axios from 'axios';
import { AuthContext } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';
import '../styles/Register.css';

const Register = () => {
  const [formData, setFormData] = useState({ name: '', email: '', password: '' });
  const [error, setError] = useState('');
  const { login } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post('http://localhost:4500/api/auth/register', formData);
      login(res.data.token);
      navigate('/');
    } catch (err) {
      setError(err.response?.data?.msg || 'Registration failed');
    }
  };

  return (
    <div className="register">
      <h2>Register</h2>
      {error && <p className="error">{error}</p>}
      <form onSubmit={handleSubmit}>
        <div>
          <label>Name:</label>
          <input type="text" name="name" value={formData.name} onChange={handleChange} required />
        </div>
        <div>
          <label>Email:</label>
          <input type="email" name="email" value={formData.email} onChange={handleChange} required />
        </div>
        <div>
          <label>Password:</label>
          <input type="password" name="password" value={formData.password} onChange={handleChange}
required />
        </div>
        <button type="submit">Register</button>
      </form>
    </div>
  );
};

export default Register;
```

7.2 Backend

adminController.js

```
const User = require('../models/User');
const Admin = require('../models/Admin');
const SearchHistory = require('../models/SearchHistory');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

// Admin authentication
exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const admin = await Admin.findOne({ email });

    if (!admin) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const isMatch = await bcrypt.compare(password, admin.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const token = jwt.sign({ id: admin._id, role: 'admin' }, process.env.JWT_SECRET, {
      expiresIn: '1d'
    });

    res.json({ token, admin: { id: admin._id, name: admin.name, email: admin.email } });
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};

// Create new user
exports.createUser = async (req, res) => {
  try {
    const { name, email, password } = req.body;

    // Check if user already exists
    let user = await User.findOne({ email });
    if (user) {
      return res.status(400).json({ message: 'User already exists' });
    }

    // Create new user
    const hashedPassword = await bcrypt.hash(password, 10);
    user = new User({
      name,
      email,
      password: hashedPassword
    });

    await user.save();
    res.status(201).json({ message: 'User created successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};

// Get all users
exports.getUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};
```

```

    const users = await User.find().select('-password');
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};

// Update user
exports.updateUser = async (req, res) => {
  try {
    const { id } = req.params;
    const { name, email, password } = req.body;

    const updateData = { name, email };

    // Only update password if provided
    if (password) {
      updateData.password = await bcrypt.hash(password, 10);
    }

    const user = await User.findByIdAndUpdate(
      id,
      updateData,
      { new: true }
    ).select('-password');

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    res.json(user);
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};

// Delete user
exports.deleteUser = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findByIdAndDelete(id);

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Also delete user's search history
    await SearchHistory.deleteMany({ userId: id });

    res.json({ message: 'User deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Server error' });
  }
};

// Get user search history
exports.getUserSearchHistory = async (req, res) => {
  try {
    const { userId } = req.params;
    console.log('Fetching search history for user:', userId);

    // First verify the user exists
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
  }
};

```



```

// Fetch search history
const history = await SearchHistory.find({ userId })
  .sort({ timestamp: -1 })
  .limit(100);

console.log('Found search history entries:', history.length);

// Format the history for better display
const formattedHistory = history.map(entry => ({
  _id: entry._id,
  user: {
    name: user.name,
    email: user.email
  },
  searchQuery: entry.searchQuery,
  timestamp: entry.timestamp,
  resultCount: entry.resultCount,
  results: entry.results
}));

console.log('Formatted history:', formattedHistory);
res.json(formattedHistory);
} catch (error) {
  console.error('Error fetching search history:', error);
  res.status(500).json({ message: 'Server error', error: error.message });
}
};

```

db.js

```

const mongoose = require('mongoose');

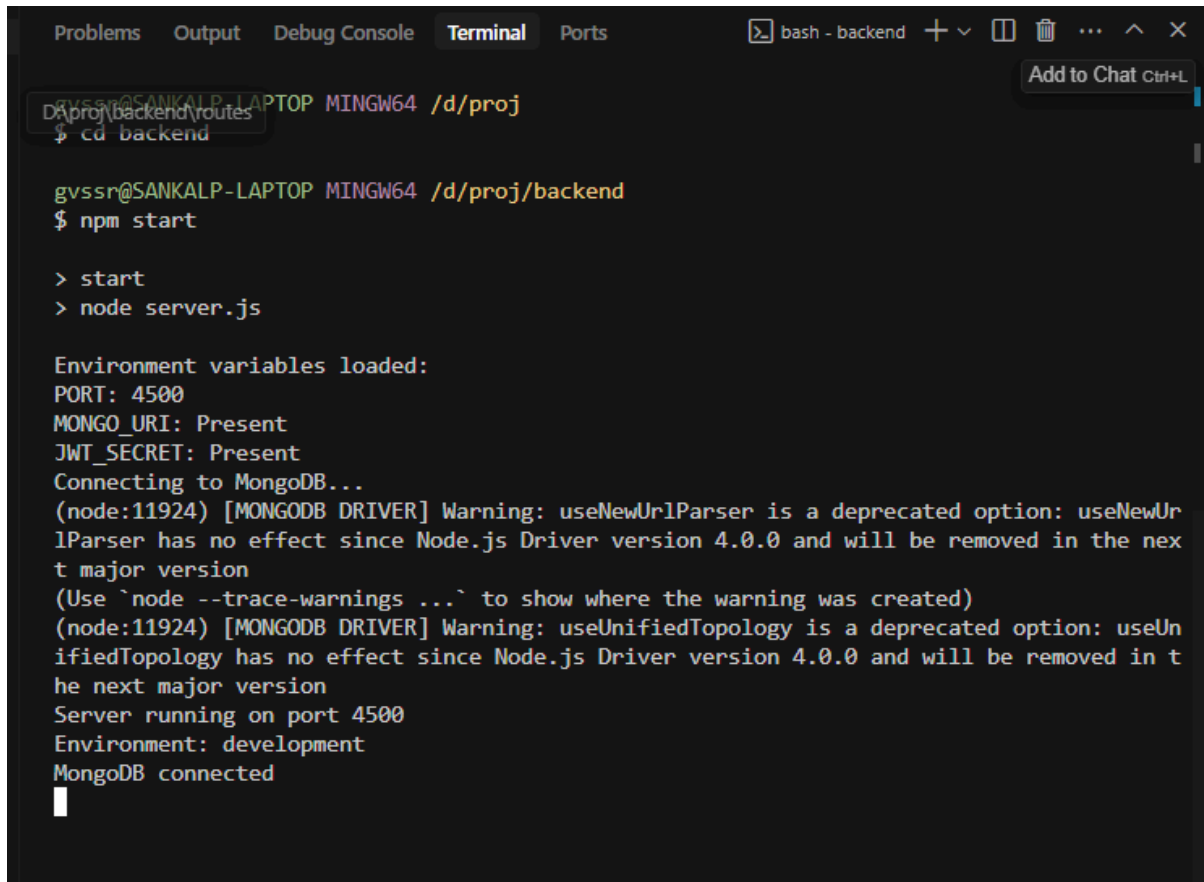
const connectDB = async () => {
  try {
    console.log('Connecting to MongoDB...');
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (error) {
    console.error('MongoDB connection error:', error);
    process.exit(1);
  }
};

module.exports = connectDB;

```

7.3 Implementation

Backend Connection Established

A terminal window titled 'bash - backend' showing the process of starting a Node.js backend. The user navigates to the backend directory and runs 'npm start'. The output shows environment variables (PORT: 4500, MONGO_URI: Present, JWT_SECRET: Present), a MongoDB connection warning, and the server starting on port 4500.

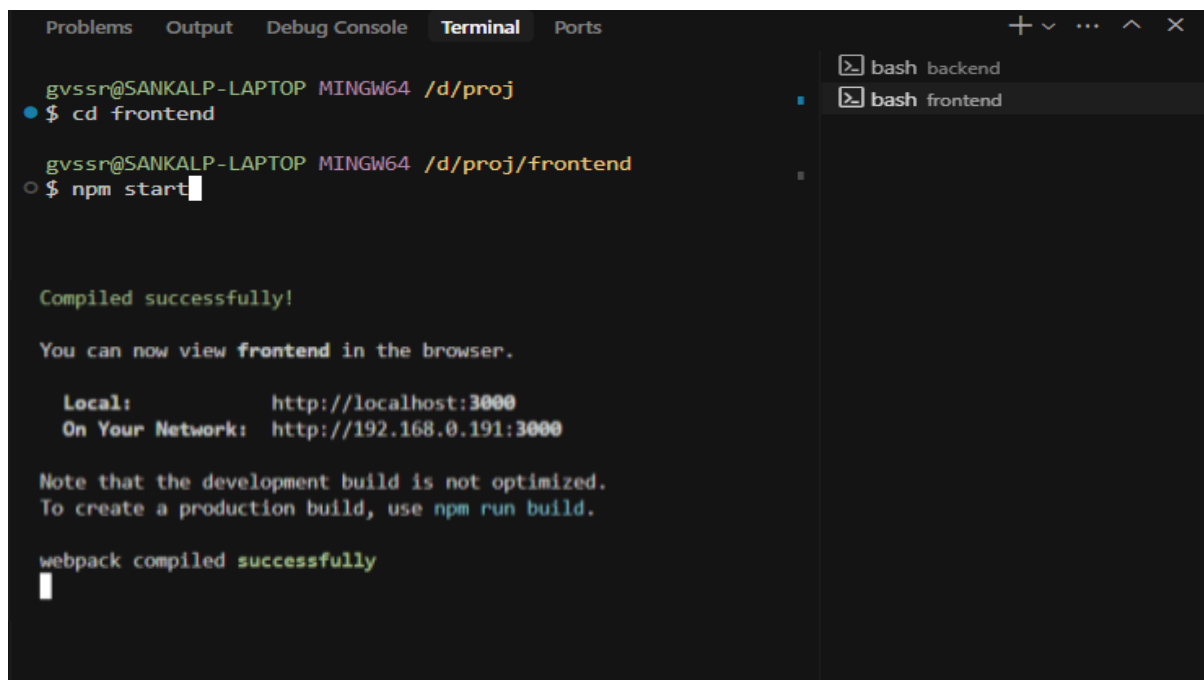
```
gvssr@SANKALP-LAPTOP MINGW64 /d/proj
D:\proj\backend\routes
$ cd backend

gvssr@SANKALP-LAPTOP MINGW64 /d/proj/backend
$ npm start

> start
> node server.js

Environment variables loaded:
PORT: 4500
MONGO_URI: Present
JWT_SECRET: Present
Connecting to MongoDB...
(node:11924) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:11924) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Server running on port 4500
Environment: development
MongoDB connected
```

Webpage Deployment (Frontend)

A terminal window titled 'bash - frontend' showing the process of starting a webpack frontend. The user navigates to the frontend directory and runs 'npm start'. The output shows a successful compilation and the server running on port 3000. A second terminal window titled 'bash - backend' is also visible in the background.

```
gvssr@SANKALP-LAPTOP MINGW64 /d/proj
$ cd frontend

gvssr@SANKALP-LAPTOP MINGW64 /d/proj/frontend
$ npm start

Compiled successfully!

You can now view frontend in the browser.

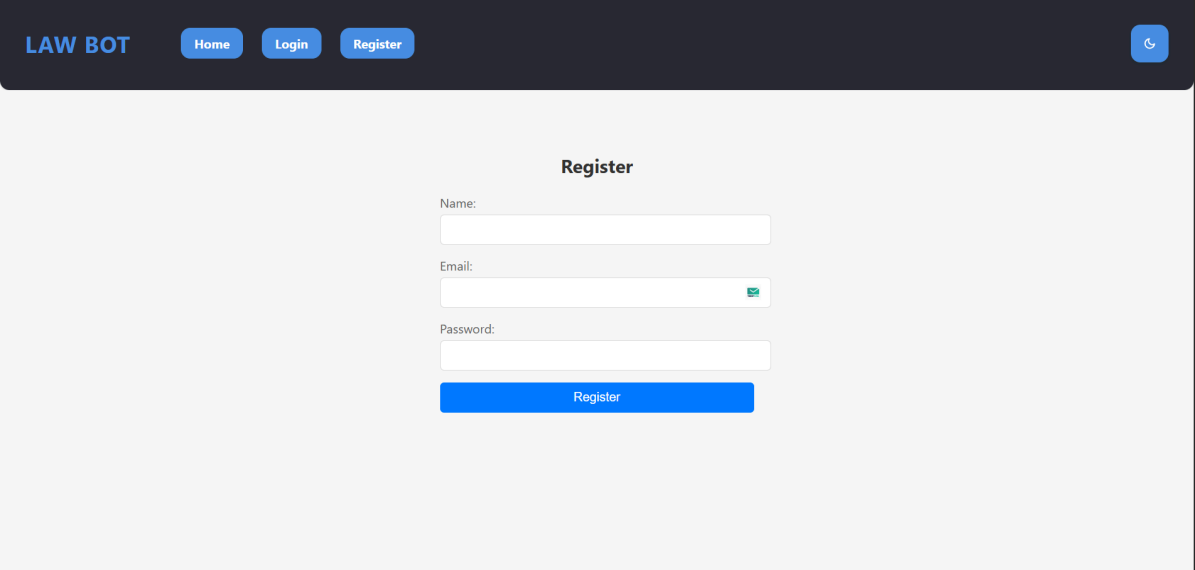
Local:      http://localhost:3000
On Your Network: http://192.168.0.191:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

7.4 Output

User Registration



A screenshot of a web browser displaying the 'Register' form. The browser's address bar shows 'http://localhost:3000/'. The page has a dark blue header with the text 'LAW BOT' and three navigation buttons: 'Home', 'Login', and 'Register'. A blue circular icon with a white refresh symbol is in the top right corner. The main content area is light gray and contains the title 'Register' in bold. Below the title are three input fields: 'Name:', 'Email:', and 'Password:'. The 'Email:' field has a small green envelope icon on its right. At the bottom of the form is a blue button labeled 'Register'.

Register

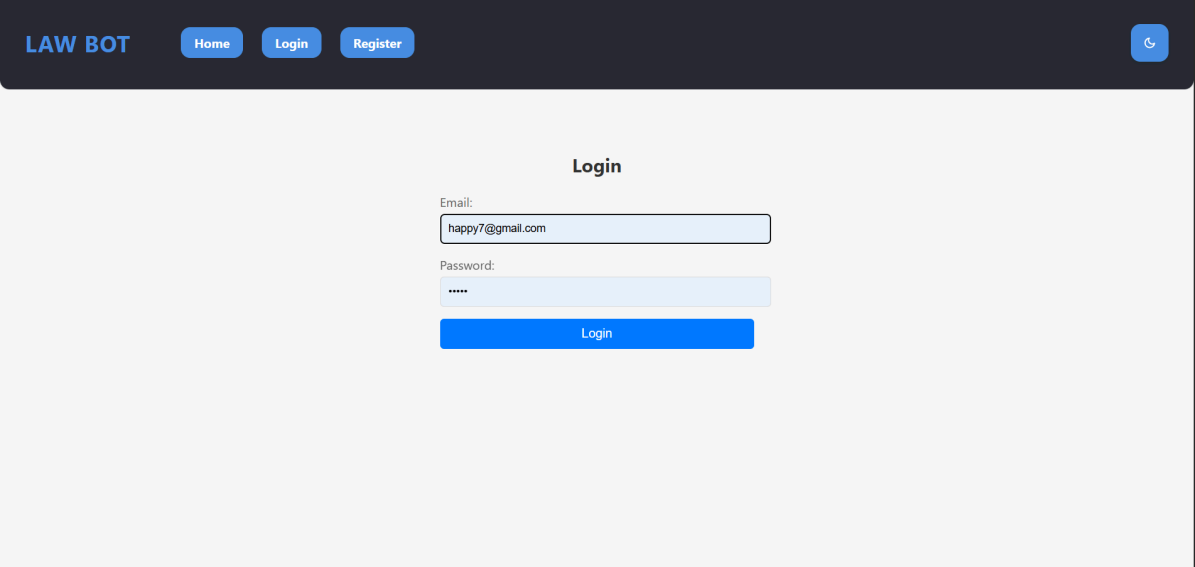
Name:

Email:

Password:

Register

User Login



A screenshot of a web browser displaying the 'Login' form. The browser's address bar shows 'http://localhost:3000/'. The page has a dark blue header with the text 'LAW BOT' and three navigation buttons: 'Home', 'Login', and 'Register'. A blue circular icon with a white refresh symbol is in the top right corner. The main content area is light gray and contains the title 'Login' in bold. Below the title are two input fields: 'Email:' and 'Password:'. The 'Email:' field contains the text 'happy7@gmail.com'. The 'Password:' field contains five asterisks. At the bottom of the form is a blue button labeled 'Login'.

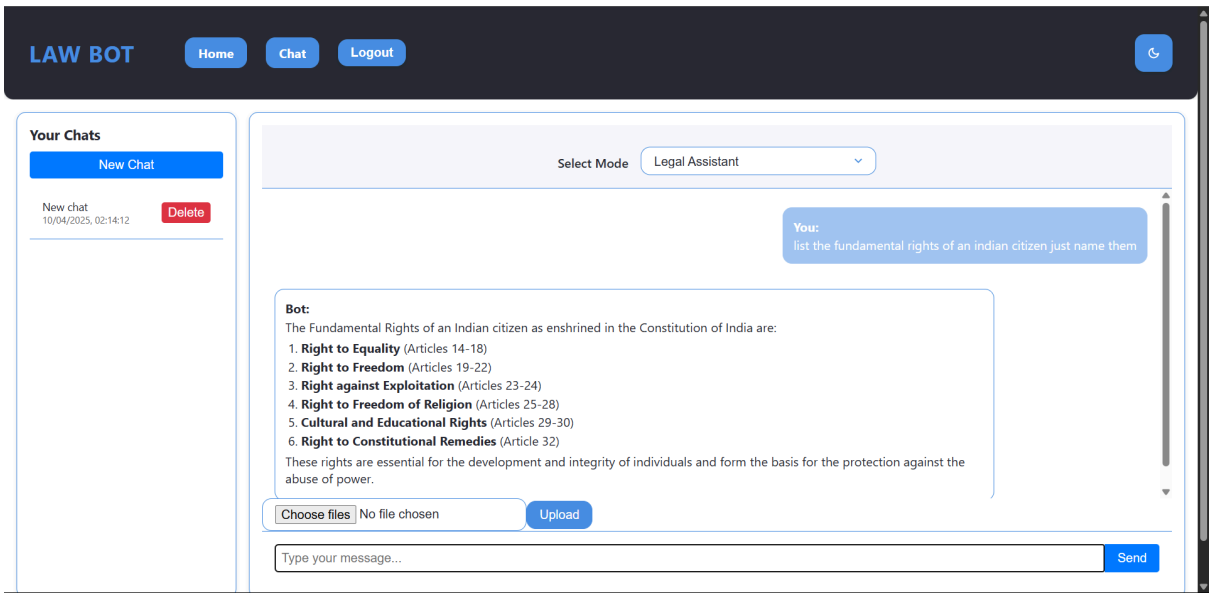
Login

Email:

Password:

Login

Chat Session



Admin Login



Admin Login

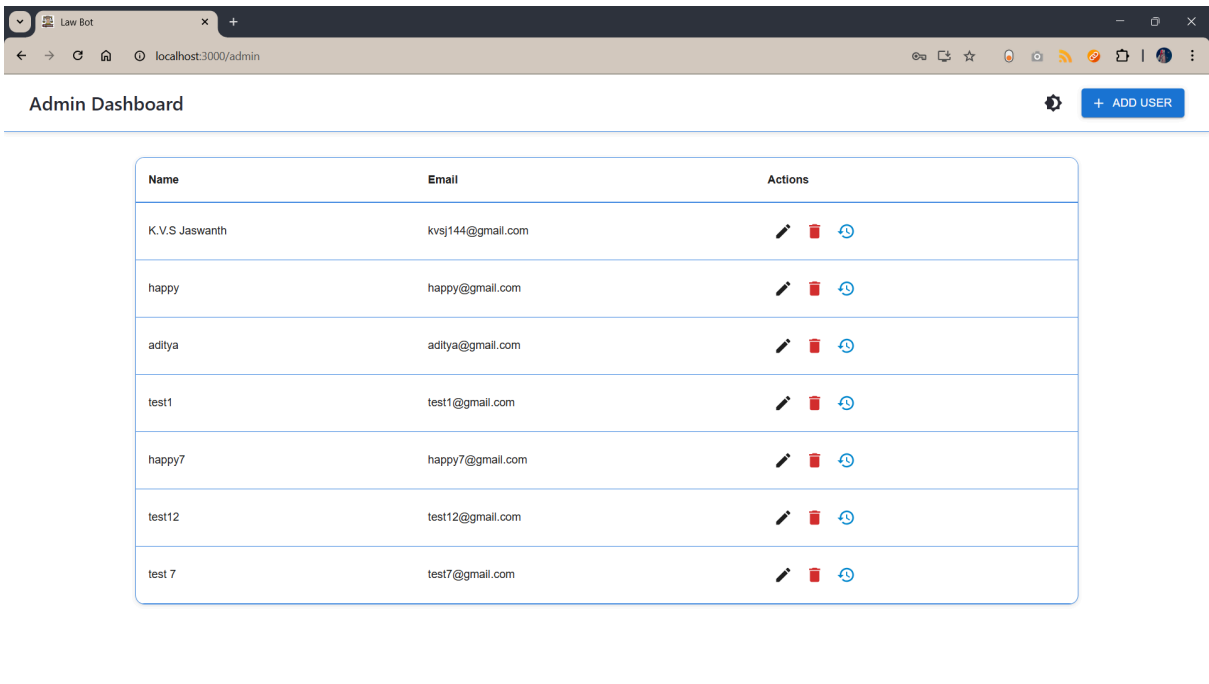
Email *

admin@gmail.com

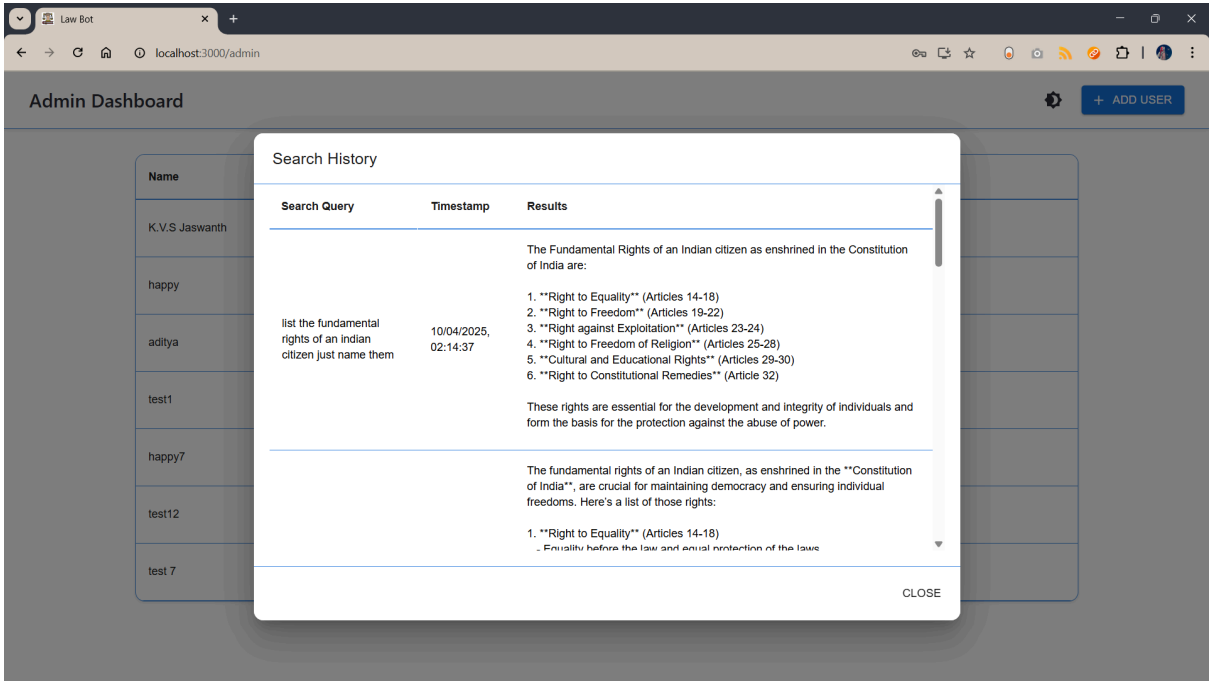
Password *

LOGIN

Admin Dashboard



Chat History



8. Conclusion and Future Scope

Conclusion

The AI Law Chatbot and Financial Advisor represent a significant advancement in simplifying the interpretation of complex legal and financial documents. By combining OCR technology for document extraction and OpenAI's language model for generating clear, user-friendly explanations, the tool offers a comprehensive solution for navigating technical content. This innovation makes intricate legal and financial terminology more accessible to users, empowering them to gain immediate, accurate insights without the need for expert consultations.

The integration of OCR technology ensures that documents are efficiently converted into machine-readable text, while OpenAI's language model breaks down the content into clear, digestible explanations. This eliminates the barriers that often come with understanding dense, jargon-heavy documents. As a result, users can make informed decisions based on the information provided, ultimately improving their ability to handle legal and financial matters independently.

The tool's automated approach reduces reliance on costly professional services, allowing individuals and businesses to save time and resources. The ease of access to these insights fosters greater confidence in users' decision-making processes, especially in areas that traditionally require specialized knowledge.

By simplifying complex content, the AI Law Chatbot and Financial Advisor offer a transformative solution, enhancing efficiency and accessibility in navigating legal and financial documents. This innovative tool exemplifies the potential of AI to democratize information, providing a more equitable way for users to interact with complex fields. Ultimately, it showcases how technology can drive positive change in traditionally challenging areas, streamlining processes and making critical information easier to digest and act upon.

Future Scope

The AI Law Chatbot and Financial Advisor system has considerable potential for future expansion, which would enhance its functionality and broaden its user base. A key area for growth is the support for additional document formats, including Word documents, Excel sheets, and scanned handwritten texts. By enabling the system to process a wider variety of document types, users would have more flexibility in uploading and interpreting different kinds of files, making the tool even more useful in diverse professional environments.

Further, the AI model could be enhanced to offer context-specific legal and financial advice that is tailored to different jurisdictions. This would allow the system to provide more relevant insights based on specific regional or national regulations, addressing the diverse legal and financial needs of users from various parts of the world. Jurisdiction-specific advice would improve the tool's precision, making it more applicable to real-world scenarios.

Another potential enhancement is multilingual support, which would extend the system's accessibility to a global audience. By allowing users to interpret documents in different languages, the AI tool could cater to non-English speaking markets, democratizing access to legal and financial information. This would be especially useful for businesses and individuals dealing with international documents or those working in multicultural environments.

Additionally, incorporating real-time collaboration features would be beneficial for teams or businesses working together on document interpretation. This would allow multiple users to interact with the same document simultaneously, making it easier for teams to review, discuss, and make decisions based on the AI-generated insights in real-time. Such collaboration tools could be particularly valuable for legal firms, financial advisors, and corporate teams managing complex projects.

Finally, integrating the system with legal and financial databases would further elevate its utility. By pulling in accurate, up-to-date information from trusted sources, the AI could provide more reliable and actionable advice. This integration would enhance the system's ability to stay current with changing laws, financial regulations, and market conditions, offering users the most precise and relevant advice available. These advancements would significantly increase the system's reach, helping a wider range of users make informed decisions with confidence.

9. References

1. Binns, R. (2018). "Understanding and Addressing the Ethical Implications of AI in Legal Systems." *Journal of Artificial Intelligence and Law*, 26(2), 129-145. (<https://link.springer.com/journal/10506>).
2. Kearns, M., & Roth, A. (2019). *The Ethical Algorithm: The Science of Socially Aware Algorithm Design*. Oxford University Press. (<https://global.oup.com/academic/product/the-ethical-algorithm-9780190948204>).
3. Dastin, J. (2020). "AI's Role in Financial Services: Transforming Document Interpretation and Automation." *Financial Times*. (<https://www.ft.com/content/d4742a4f-14b7-406d-9f2e-d76b82c07e5d>).
4. Müller, V. C. (2020). *AI in Law and Legal Practice: The Role of AI in Legal Decision-Making*. Springer. (<https://www.springer.com/gp/book/9783030241531>)
5. Beck, J., & Corrigan, D. (2021). "Expanding AI Capabilities in Document Management: From OCR to Multilingual Interpretation." *Journal of Document Management Technology*, 18(1), 45-60. (<https://www.journals.elsevier.com/journal-of-document-management>)
6. Agarwal, A., & Venkatraman, S. (2021). "Collaboration Tools and AI: Improving Team Productivity in Financial and Legal Sectors." *Journal of Business Technology*, 33(4), 59-75. (<https://www.journals.elsevier.com/journal-of-business-research>).
7. KPMG (2022). "AI and Financial Services: How Artificial Intelligence is Revolutionizing the Industry." KPMG Global Insights. (<https://home.kpmg/xx/en/home/insights/2021/02/ai-and-financial-services.html>)