Group 15

LegalAI

Legal Information Retrieval System

Software Architecture Document

Version 1.1

Revision History

Date	Version	Description	Author
02/08/2025	1.0	Initial SAD	Kisara Kodithuwakku
03/08/2025	1.1	Update diagram descriptions Replace Gemini with generic LLM API	Kisara Kodithuwakku

Table of Contents

1. Introduction 5
   - 1.1 Purpose 5
   - 1.2 Scope 5
   - 1.3 Definitions, Acronyms, and Abbreviations 5
   - 1.4 References 5
   - 1.5 Overview 6

2. Architectural Representation 7
   - 2.1 Use-Case View 7
   - 2.2 Logical View 7
   - 2.3 Process View 7
   - 2.4 Deployment View 7
   - 2.5 Implementation View 7
   - 2.6 Data View 7

3. Architectural Goals and Constraints 8
   - 3.1 Goals 8
   - 3.2 Constraints 9

4. Use-Case View 10
   - 4.1 Use-Case Realizations 10

5. Logical View 12
   - 5.1 Overview 12
   - 5.2 Architecturally Significant Design Packages 12

6. Process View 14
   - 6.1 Activity Diagram 14
   - 6.2 Sequence Diagram 15

7. Deployment View 16
8. Implementation View 18
   - 8.1 Overview 18
   - 8.2 Package Diagram 18
   - 8.3 Layers 18

9. Data View 20
   - 9.1 Stored Data 20

10. Size and Performance 21
10.1 Dimensioning 21
10.2 Performance Constraints 21
11. Quality 22
11.1 Extensibility 22
11.2 Reliability 22
11.3 Portability 22
11.4 Security & Privacy 22
11.5 Accessibility 22
12. References 23

1. Introduction

1.1 Purpose

This Software Architecture Document (SAD) provides a comprehensive architectural overview of the Simple Legal Information Retrieval System for Sri Lanka. It describes the system's structure, major components, interactions, and design decisions. The SAD is intended for developers, stakeholders, and maintainers to understand, implement, and evolve the system.

1.2 Scope

This document applies to the web-based legal information retrieval system, as outlined in the SRS. It covers all architectural aspects, including functional modules (chatbot, search engine, recommendation system), integration with external APIs, data storage, and user interface.

1.3 Definitions, Acronyms, and Abbreviations

- RAG (Retrieval-Augmented Generation): Combines document retrieval and generative AI for answering queries.
- LLM (Large Language Model): Generative AI model for generating human-like text.
- Vector Database: Stores vector embeddings for semantic search.

1.4 References

- Software Requirements Specification (SRS), v1.0, 02/08/2025

Resources

- lk_legal_docs dataset: https://github.com/nuuuwan/lk_legal_docs
- FAISS Documentation: https://python.langchain.com/docs/integrations/vectorstores/faiss/
- LaBSE: https://huggingface.co/sentence-transformers/LaBSE
- Legal-BERT: https://huggingface.co/nlpaueb/legal-bert-base-uncased

Similar Projects

- LankaLaw Platform: https://lankalaw.net/
- AIPazz Legal AI Tool: https://www.aipazz.com

Papers

- Large Language Models: A Survey: https://arxiv.org/abs/2402.06196
- Retrieval-Augmented Generation for Large Language Models: A Survey: https://arxiv.org/abs/2312.10997
- Mistral 7B: https://arxiv.org/abs/2310.06825

Tools

- Draw.io: https://app.diagrams.net/
- Lucid Chart: https://lucid.app/lucidchart/

1.5 Overview

The SAD is organized into architectural views: use-case, logical, process, deployment, implementation, data, performance, and quality. Each section details the relevant aspects and includes placeholders for diagrams and further documentation.

2. Architectural Representation

The architecture of the Simple Legal Information Retrieval System is described using several complementary views, each providing a different perspective on the system's structure and behavior. This multi-view approach ensures that all stakeholders understand the system at both high and low levels of abstraction.

2.1 Use-Case View

- Focuses on the interactions between users (citizens, legal professionals) and the system.
- Identifies the main functionalities provided, such as legal queries, document search, recommendations, and updates.
- Provides context for requirements traceability and validation.

2.2 Logical View

- Describes the static structure of the system, including major packages, modules, and classes.
- Shows how responsibilities are distributed and how components interact internally.
- Supports maintainability and extensibility by clarifying boundaries and dependencies.

2.3 Process View

- Illustrates the dynamic behavior of the system, including concurrent processes and workflows.
- Details how requests are handled, how data flows, and how asynchronous operations (e.g., updates) are managed.
- Ensures reliability and performance by mapping out critical paths and interactions.

2.4 Deployment View

- Maps software components to physical infrastructure (servers, databases, cloud services).
- Addresses scalability, availability, and integration with external APIs (LLM Service, GitHub).
- Guides DevOps and infrastructure planning.

2.5 Implementation View

- Details the organization of code into layers and subsystems.
- Clarifies the rules for component inclusion and the interfaces between layers.
- Facilitates modular development and future enhancements.

2.6 Data View

- Describes the persistent storage model, including document embeddings, user data, and legal document metadata.
- Ensures data integrity, security, and efficient retrieval.

3. Architectural Goals and Constraints

3.1 Goals

The architecture is designed to achieve the following key goals, each of which directly influences design decisions and system quality:

Reliability

- The system must consistently provide accurate and relevant legal information. This is achieved through robust error handling, redundancy in critical components, and automated testing. For example, if a search fails, the system should notify the user and log the error for review.
- High availability is ensured by deploying the system on scalable cloud infrastructure that maintains regular backups of user data.

Performance

- Fast response times are essential for user satisfaction. The architecture leverages efficient data retrieval (e.g., vector database for semantic search) and optimized backend processes to ensure search results are returned within 3 seconds and chatbot responses within 5 seconds.
- Caching strategies and asynchronous processing are used to minimize latency and maximize throughput.

Scalability

- The system must accommodate growth in user base, data volume, and feature set. Modular design and cloud-based deployment allow for horizontal scaling (adding more servers) and vertical scaling (upgrading resources).
- The architecture supports incremental feature addition, such as new languages or recommendation algorithms, without major refactoring.

Security & Privacy

- User data is protected through encryption at rest and in transit, secure authentication mechanisms, and role-based access control.
- Compliance with legal and ethical standards, is maintained by minimizing data storage and regularly auditing access logs.

Usability Accessibility

- The user interface is designed to be intuitive and accessible to a wide range of users, including support for Sinhala and English.

Maintainability & Extensibility

- Clear separation of concerns and well-defined interfaces between components facilitate easy updates, bug fixes, and addition of new features.

3.2 Constraints

The architecture must operate within the following constraints, which shape the technical and operational boundaries of the system:

Platform Compatibility:

- The system must be accessible via modern desktop and mobile browsers, ensuring a consistent experience across devices and operating systems.
- Responsive design and cross-browser testing are required to meet this constraint.

Integration Requirements:

- Seamless integration with external services is mandatory, including LLM API for AI-powered responses, vector database for semantic search, and GitHub for legal document updates.
- The architecture must accommodate changes in third-party APIs and provide fallback mechanisms if external services are unavailable.

Technology Choices:

- Preference is given to open-source technologies to minimize costs, avoid vendor lock-in, and leverage community support.
- Selected technologies must be well-documented, actively maintained, and compatible with the system's requirements.

Legal & Ethical Compliance:

- The system must adhere to data protection laws, content reuse guidelines, and AI safety practices. For example, only public domain legal documents are used, and user data is handled according to privacy regulations.
- Regular legal reviews and compliance audits are part of the maintenance process.

Resource Limitations:

- Efficient use of computational resources is required, especially for AI inference and data storage, to control operational costs and ensure sustainability.
- The architecture supports monitoring and scaling to optimize resource usage based on demand.

4. Use-Case View

4.1 Use-Case Realizations

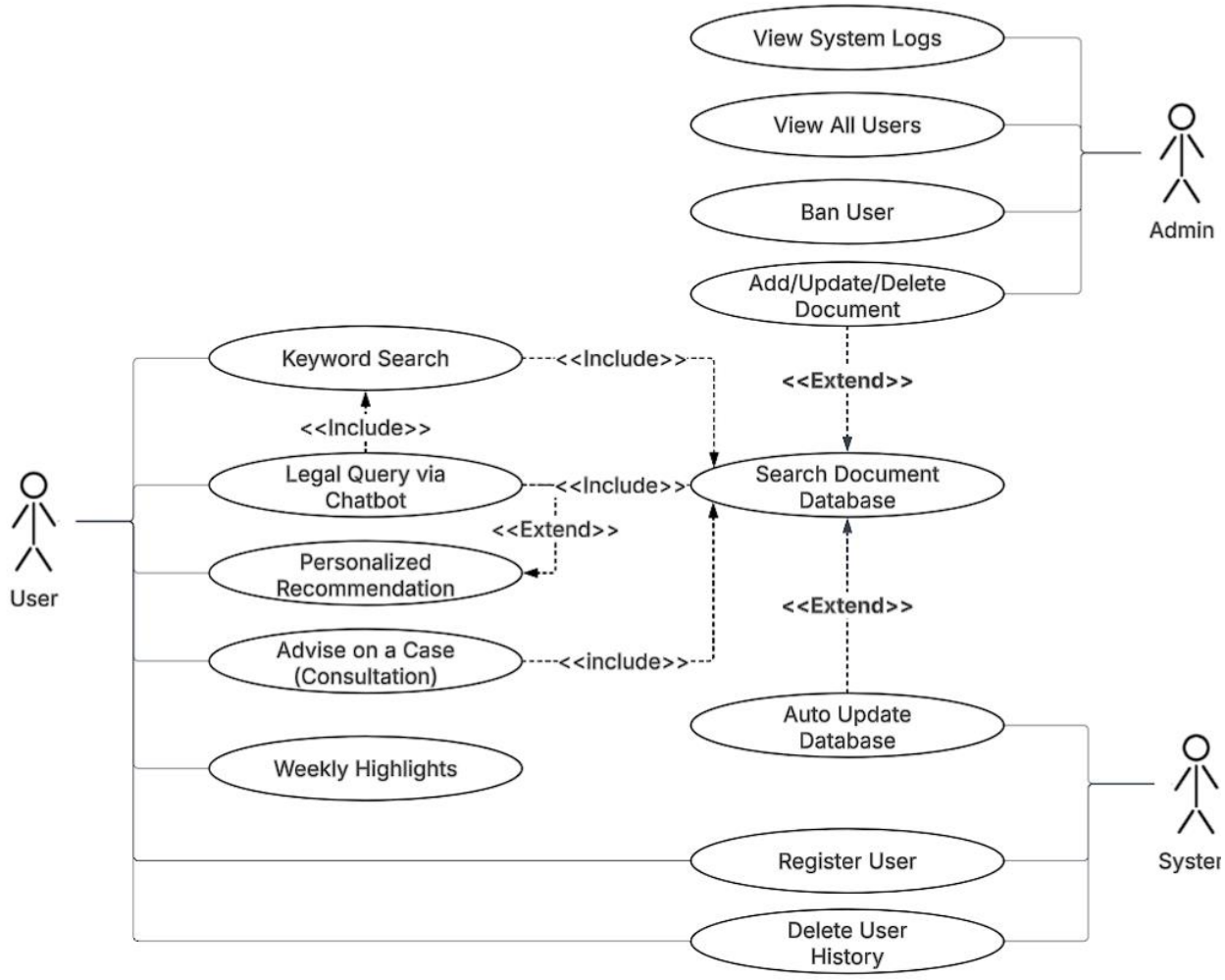

Figure 1: Use Case Diagram

Use Case: Legal Query via Chatbot

Actor: User

Description: User asks a legal question; chatbot retrieves relevant documents and generates a response.

Preconditions: User is authenticated (for personalized features).

Main Flow:

1. User enters query.
2. System retrieves relevant documents from vector database.
3. LLM generates response.
4. Response displayed to user.

Success: User receives accurate legal information.
Failure: User receives error message or suggestion to rephrase.

Use Case: Legal Document Search
Actor: User
Description: User searches for legal documents in Sinhala or English.
Preconditions: None
Main Flow:

1. User enters search query.
2. System performs lexical and semantic search (using embeddings) in parallel.
3. Results displayed.

Success: Relevant documents shown.
Failure: Not found message.

Use Case: Personalized Recommendation
Actor: User
Description: System recommends documents based on user history.
Preconditions: User has search or query history.
Main Flow:

1. System analyzes user history.
2. System suggests related documents.

Success: Recommendations displayed.
Failure: No recommendations available.

Use Case: Automatic Legal Document Update
Actor: System
Description: System fetches new documents from GitHub and updates database.
Preconditions: GitHub connection and database online.
Main Flow:

1. Scheduled job checks for updates.
2. New documents fetched, indexed and embedded.
3. Update logged.

Success: Database updated, users notified.
Failure: Error logged, alert sent.

5. Logical View

5.1 Overview

- User Interface: Handles user interactions (search, chatbot, recommendations).
- Search Engine: Manages semantic search and multilingual queries.
- Chatbot Module: Integrates LLM and document retrieval.
- Recommendation Engine: Tracks user history and generates suggestions.
- Data Access Layer: Interfaces with vector database and user data storage.
- Update Manager: Automates legal document updates from GitHub.

5.2 Architecturally Significant Design Packages

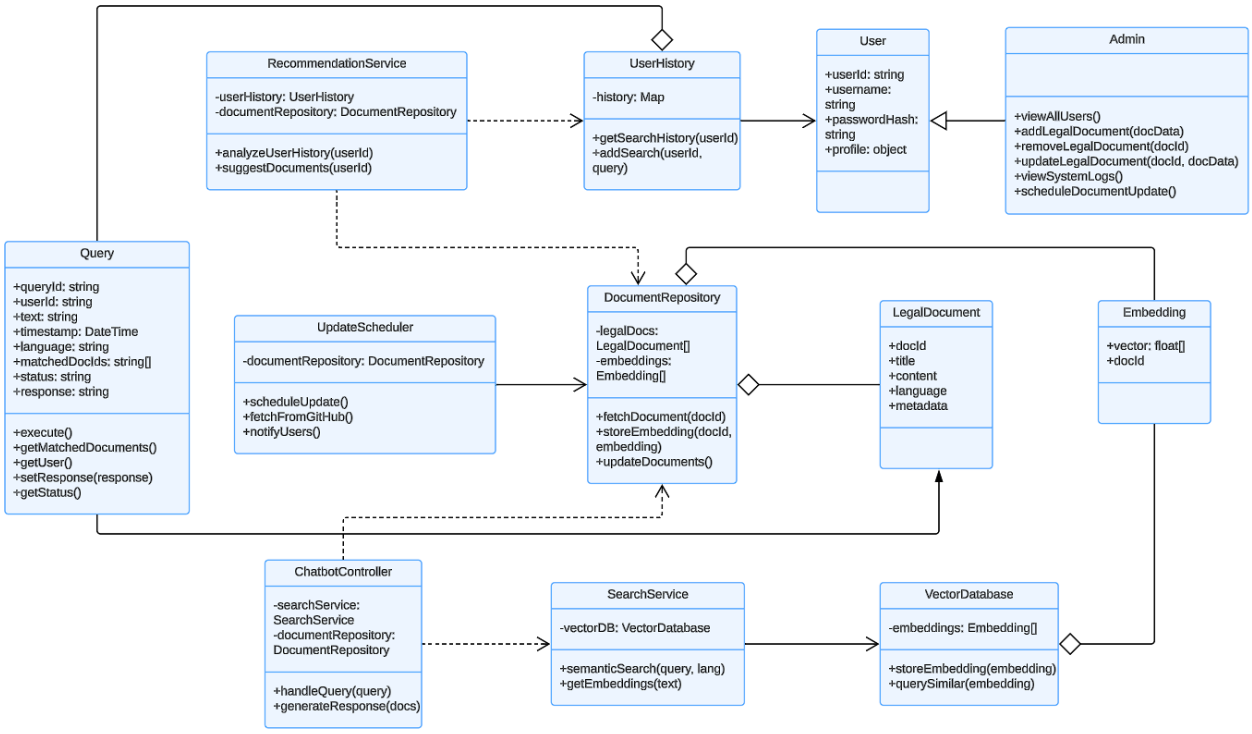

Figure 2: Class Diagram

Model Components

- User:
  Represents a system user, including authentication information, profile data, and user ID. Interacts with UserHistory for tracking searches and feedback, and is referenced by other components for personalized features and access control.

- UserHistory:
  Tracks user search history and queries. Provides methods for retrieving and updating user search records. Used by RecommendationService to personalize recommendations. Tracks multiple Query objects for each user.
- Query:
  Represents a user's search or chatbot query. Contains attributes such as queryId, userId, text, timestamp, language, matchedDocIds, status, and response. Provides methods for executing the query, retrieving matched documents, and managing query status and response.
- LegalDocument:
  Represents a legal document, including its ID, title, content, language, and metadata. Managed by DocumentRepository.
- Embedding:
  Represents a vector embedding associated with a legal document. Used for semantic similarity search and managed by DocumentRepository and VectorDatabase.

Data Management

- DocumentRepository:
  Manages access to legal documents and their embeddings. Responsible for fetching, storing, and updating documents and embeddings. Interfaces with LegalDocument and Embedding classes to maintain document metadata and vector representations.
- VectorDatabase:
  Stores and manages document embeddings for semantic search. Provides methods for storing embeddings and querying similar vectors. Used by SearchService for fast retrieval.

Controllers & Services

- ChatbotController:
  Handles user queries, interacts with the SearchService to retrieve relevant documents, and generates responses using the LLM. Maintains references to SearchService and DocumentRepository for document access and search operations.
- SearchService:
  Handles semantic search operations, including multilingual queries. Interfaces with the VectorDatabase to retrieve document embeddings and perform similarity searches. Provides methods for embedding generation and search.
- RecommendationService:
  Analyzes user search history and suggests related legal documents. Maintains references to UserHistory and DocumentRepository to access user data and documents. Provides methods for analyzing history and generating recommendations.
- UpdateScheduler:
  Automates the process of fetching new legal documents from external sources (e.g., GitHub), updating the repository, and notifying users. Maintains a reference to DocumentRepository for document updates.

6. Process View

The system consists of the following main processes:

- User Request Handling: Processes user queries and search requests.
- Document Retrieval: Fetches and embeds legal documents.
- Response Generation: Uses LLM to generate answers.
- Recommendation Generation: Computes personalized suggestions.
- Update Process: Automates document updates and notifications.

6.1 Activity Diagram

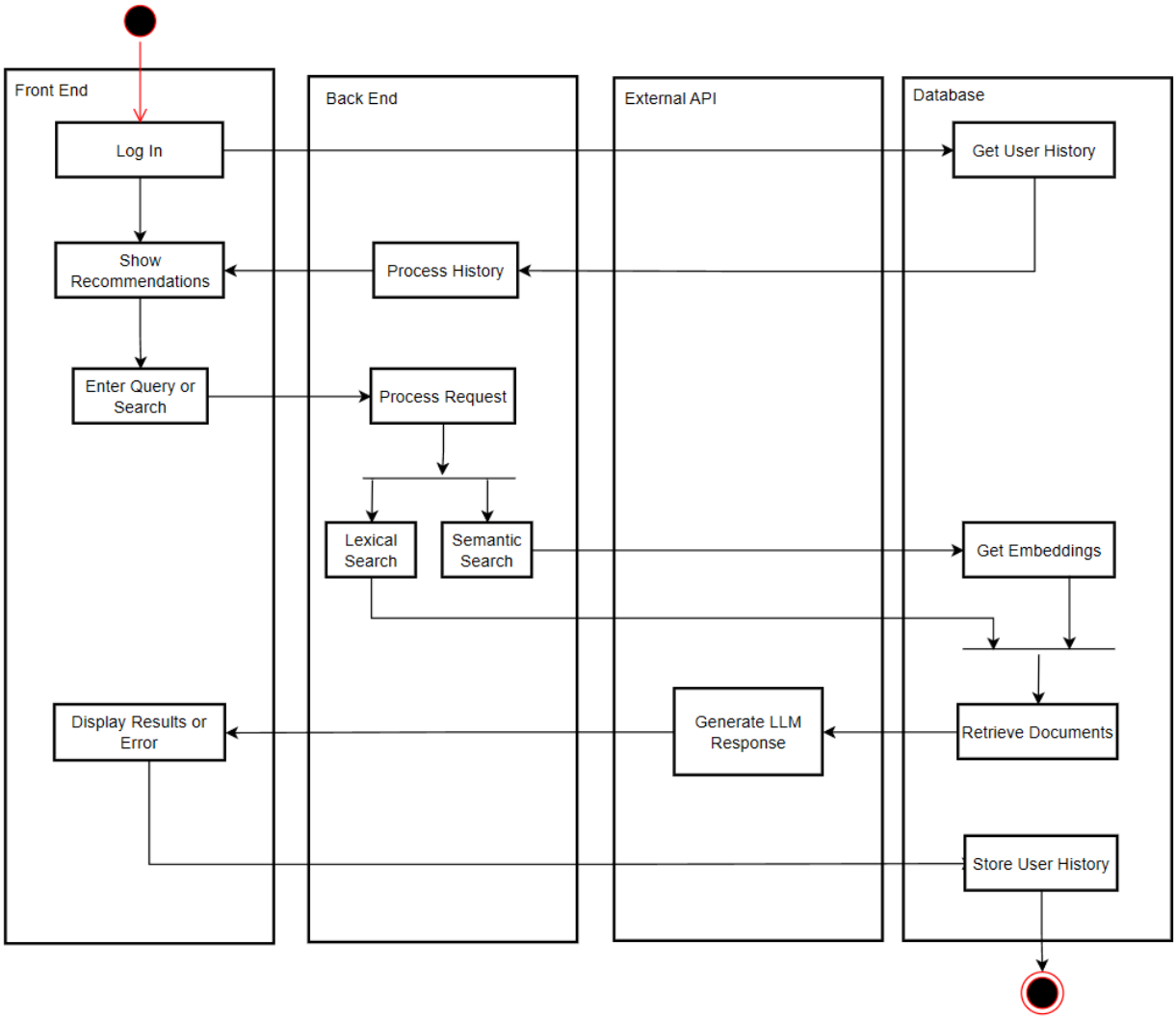

Figure 3: Activity Diagram

Figure 3 describes the main workflows of the system, including user request handling, document retrieval, response generation and recommendation. It shows how user input flows from the UI to the backend, which interacts with databases and external APIs to process queries, retrieve documents, generate responses and analyze user history.
Swimlanes represent major system components (Frontend, Backend, Database, External APIs), highlighting parallel and sequential activities and the interactions between them.

6.2 Sequence Diagram

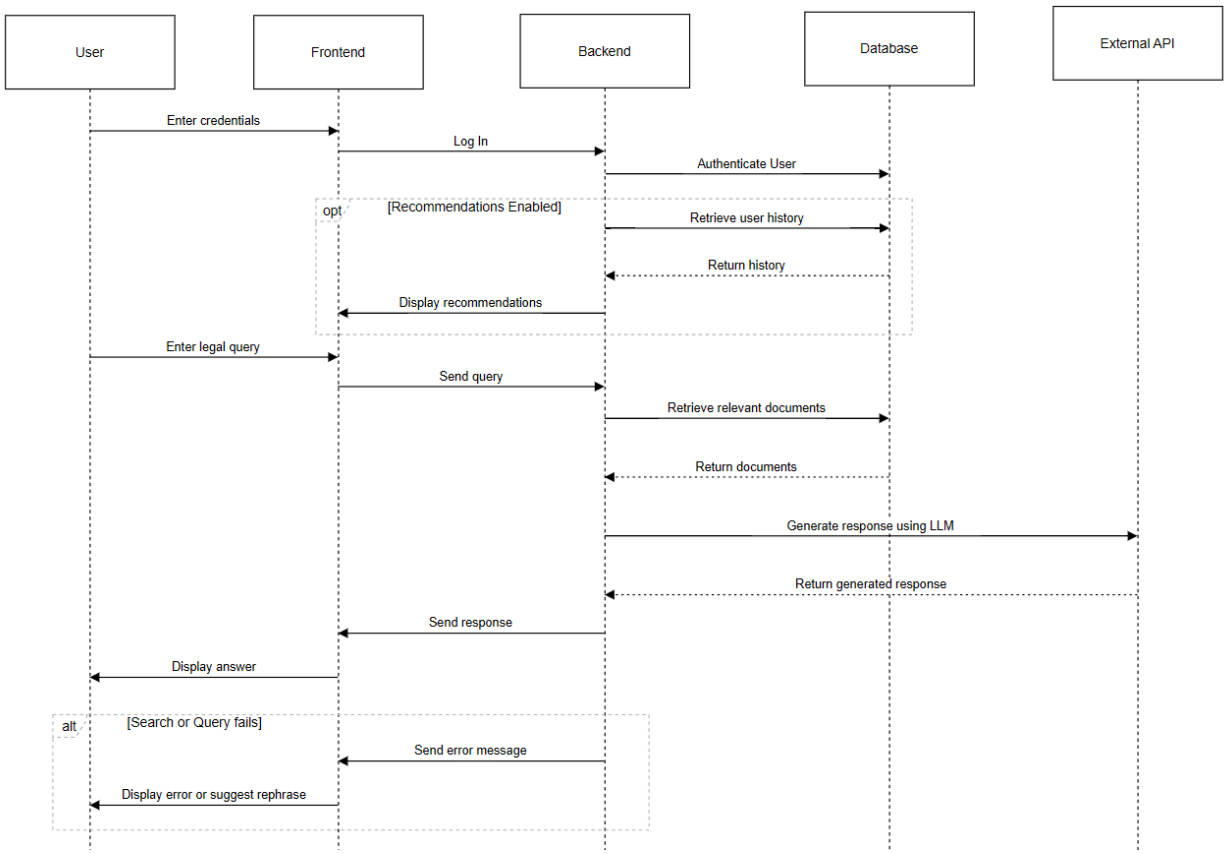

Figure 4: Sequence Diagram

Figure 4 shows a typical scenario for a legal query via the chatbot. It details the step-by-step interactions between the user, frontend UI, backend, database, and LLM API.
The diagram shows how a user submits a query, the UI forwards it to the backend, which retrieves relevant documents from the vector database, generates a response using the LLM API, and returns the answer to the user.
It also includes flows for recommendations and error handling, illustrating the order and dependencies of system operations during query processing.

7. Deployment View

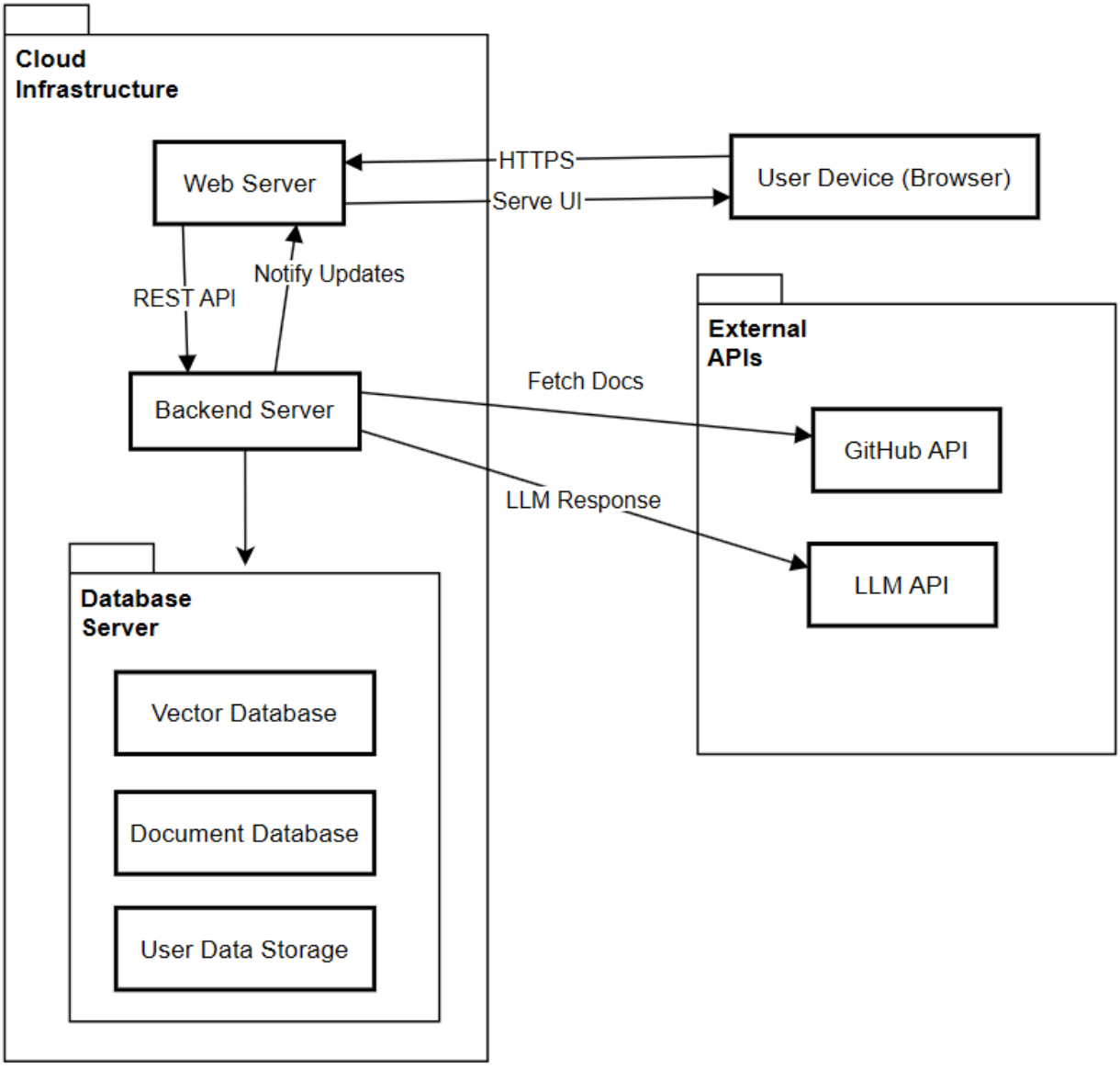

Figure 5: Deployment Diagram

User Device (Browser):

- Users access the system via modern browsers. The UI is served over secure HTTP/HTTPS connections.

Web Server:

- Hosts the user interface and exposes RESTful API endpoints for client communication.
- Handles incoming HTTP/HTTPS requests from user devices and forwards them to the application server.

- Serves static assets and UI components.

Application Server:

- Implements backend business logic, including search, chatbot, recommendations, and update processes.
- Communicates with the vector database server to query and store document embeddings.
- Integrates with external APIs (Gemini for LLM responses, GitHub for legal document updates).
- Sends notifications and updates to the web server for user display.

Vector Database:

- Stores vector embeddings for semantic search and document retrieval.
- Optimized for fast similarity queries and secure data storage.

Document Database:

- Stores legal documents and their metadata.
- Provides efficient retrieval and update operations for legal content.
- Managed by the Application Server for document access, search, and updates.

User Data Storage:

- Stores user profiles, authentication information, and search history.
- Supports personalized features such as recommendations and feedback tracking.
- Ensures secure storage and privacy of user data, accessible by the Application Server.

External APIs:

- LLM API: Provides LLM-powered responses for chatbot queries. (e.g., Gemini, OpenAI, or Hosted model with exposed API endpoints)
- GitHub API: Supplies new legal documents and updates via automated fetches.

8. Implementation View

8.1 Overview

The system follows a Layered Architecture, organized into four layers:

- Presentation Layer: User interface components.
- Application Layer: Business logic (search, chatbot, recommendations).
- Data Layer: Data access and storage (vector database, user data).
- Integration Layer: External API connectors (LLM API, GitHub).

8.2 Package Diagram

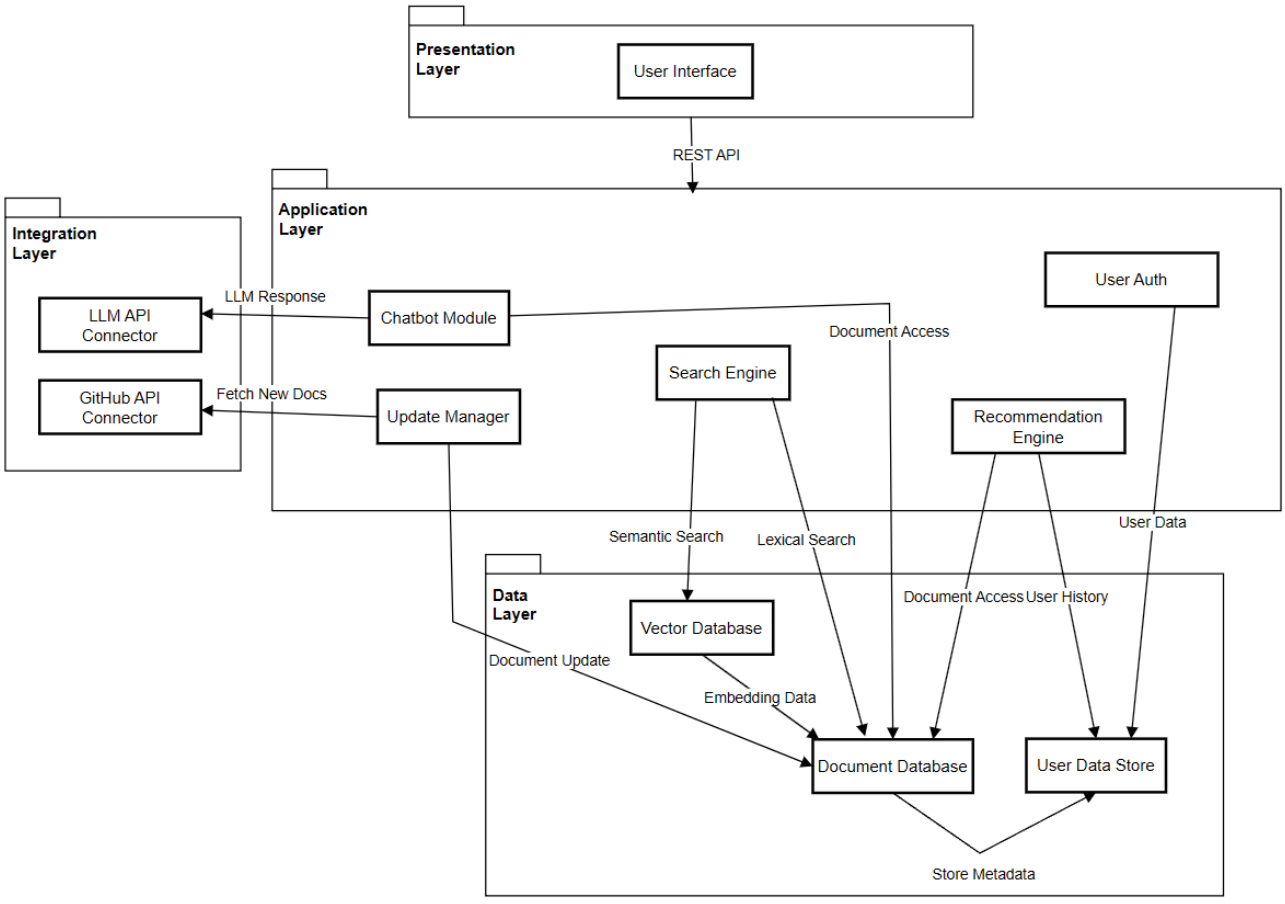

Figure 6: Package Diagram

8.3 Layers

Presentation Layer:

- Contains the User Interface (UI) components, responsible for handling user interactions, displaying search results, chatbot responses, recommendations, and notifications.

- Communicates with the backend via RESTful APIs.

Application Layer:

- Implements core business logic, including the Chatbot Module (for legal queries), Search Engine (for semantic and multilingual search), Recommendation Engine (for personalized suggestions), and Update Manager (for automated document updates).
- Coordinates requests between the UI and data/integration layers.

Data Layer:

- Manages persistent storage and retrieval of data, including the Vector Database (for embeddings), User Data Store (for search history and feedback), and Document Repository (for legal documents and metadata).
- Ensures data integrity, security, and efficient access for application logic.

Integration Layer:

- Handles connections to external services, including the LLM API (for LLM-powered responses) and GitHub API (for legal document updates).
- Provides standardized interfaces for the application layer to interact with third-party APIs.

Layer Interactions:

- The Presentation Layer interacts with the Application Layer via RESTful APIs.
- The Application Layer coordinates with the Data Layer for storage and retrieval, and with the Integration Layer for external API calls.
- The Data Layer provides secure, efficient access to embeddings, user data, and documents.

9. Data View

9.1 Stored Data

- LegalDocument: Contains metadata and content for each legal document, including document ID, title, language, and other attributes.
- Embedding: Stores vector representations for semantic search, linked to specific legal documents.
- User: Represents system users, including authentication information and profile data.
- UserHistory: Tracks user search history and feedback for personalized recommendations.

The vector database is optimized for fast similarity search. User and UserHistory data is stored securely with encryption, and is protected according to privacy and security requirements.

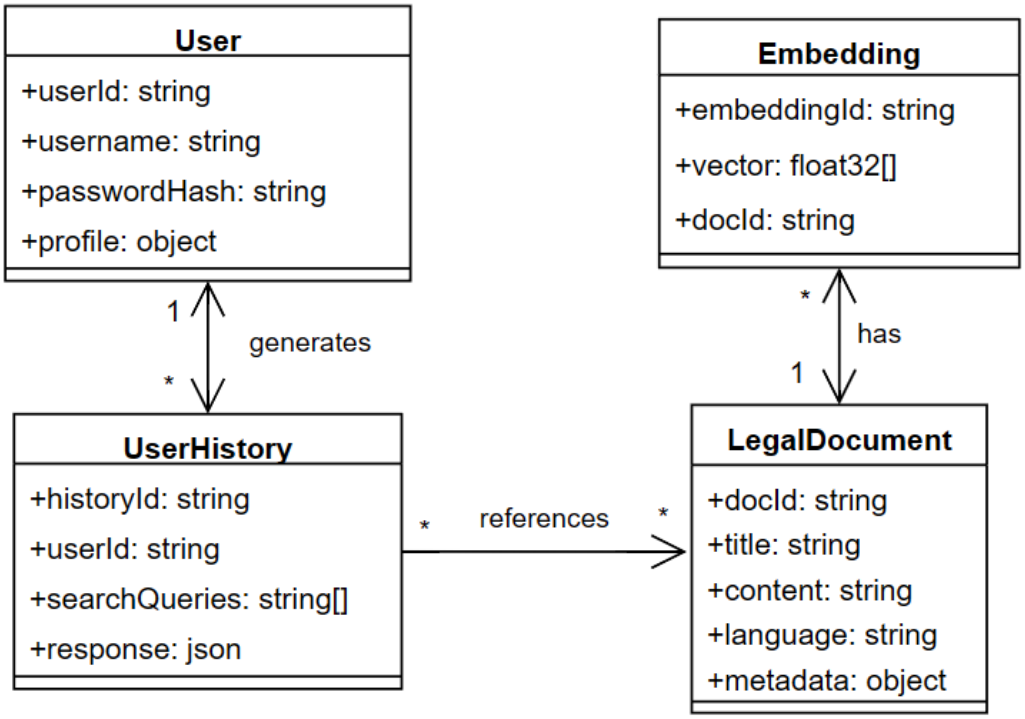

Figure 7: Data Model View

10. Size and Performance

10.1 Dimensioning

- System designed to handle thousands of documents and concurrent users.
- Embedding storage optimized for fast retrieval.

10.2 Performance Constraints

- Search results returned within 3 seconds.
- Chatbot responses generated within 5 seconds, or 0.1 seconds per word when streaming.
- High availability (99% uptime, excluding maintenance with at least 7 day prior notice).

11. Quality

11.1 Extensibility

- Modular architecture allows for easy addition of new features (e.g., different language models, new recommendation algorithms, more document types).

11.2 Reliability

- Robust error handling and automated testing ensure consistent operation.

11.3 Portability

- Web-based system compatible with modern browsers and mobile devices.

11.4 Security & Privacy

- User credentials encrypted and access controlled.
- Compliance with data protection laws.

11.5 Accessibility

- Follows accessibility standards for UI and content.

12. References

- Software Requirements Specification (SRS), v1.0, 02/08/2025

Resources

- lk_legal_docs dataset: https://github.com/nuuuwan/lk_legal_docs
- FAISS Documentation: https://python.langchain.com/docs/integrations/vectorstores/faiss/
- LaBSE: https://huggingface.co/sentence-transformers/LaBSE
- Legal-BERT: https://huggingface.co/nlpaueb/legal-bert-base-uncased

Similar Projects

- LankaLaw Platform: https://lankalaw.net/
- AIPazz Legal AI Tool: https://www.aipazz.com

Papers

- Large Language Models: A Survey: https://arxiv.org/abs/2402.06196
- Retrieval-Augmented Generation for Large Language Models: A Survey: https://arxiv.org/abs/2312.10997
- Mistral 7B: https://arxiv.org/abs/2310.06825

Tools

- Draw.io: https://app.diagrams.net/
- Lucid Chart: https://lucid.app/lucidchart/