Project - High Level Design On Legal Case Management System with CI/CD Pipeline

Course Name: DevOps

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	Jogendar Das Bairagi	EN22ME304045
2.	Roshan Bankar	EN22EL301072
3.	Vivek Gangrade	EN22CS3011098
4.	Maahi Maheshwari	EN22ME304057
5.	Shubhangi Prajapat	EN22ME304094

Group Name:04D10

Project Number:DO-17

Industry Mentor Name:

University Mentor Name:

Academic Year:2025-26

MEDICAPS UNIVERSITY

Datagami
"Lead Digital Technology"

Table of Contents

1. Introduction

The Legal Assistance Web Application is a web-based platform designed to provide users with easy access to basic legal information and assistance. The application helps users understand legal procedures, laws, and rights through a simple and user-friendly interface. It aims to reduce the gap between common people and legal knowledge by providing structured legal content online.
The project is developed using modern web technologies and focuses on simplicity, accessibility, and responsiveness. It is especially useful for individuals who need preliminary legal guidance before consulting a legal professional.

1.1 Scope of the Document

This document provides a **High-Level Design (HLD)** of the Legal Assistance Web Application.
It explains:

- Overall system architecture
- Application design and workflow
- Data handling and security aspects
- Non-functional requirements

This document does **not** include low-level code implementation details.

1.2 Intended Audience

- Project evaluators and faculty members
- Students and developers
- Industry mentors
- Anyone interested in understanding the system design of the application

1.3 System Overview

The Legal Assistance Web Application is a client-side web application that runs on a browser. Users can access legal content without installing any software. The system emphasizes simplicity, speed, and accessibility.
The system is a web-based application that allows users to:

- Access legal information
- Navigate through different legal categories
- View legal content in a structured manner

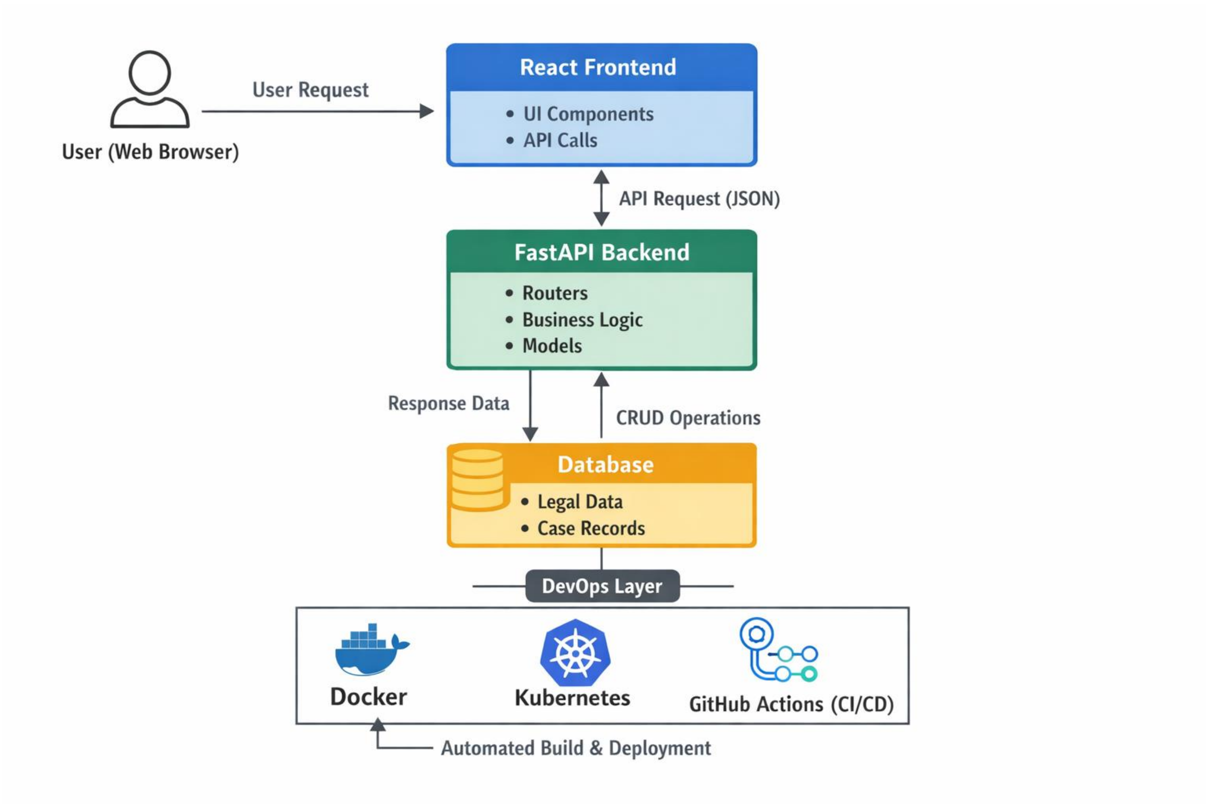
The application runs on a browser and does not require any special software installation.

2. System Design

This project is a web-based application designed to manage legal cases efficiently. Users such as lawyers or admins can create, update, view, and delete case records, manage documents, and track case status. The system provides a structured and centralized way to handle legal information digitally.

Tech Stack

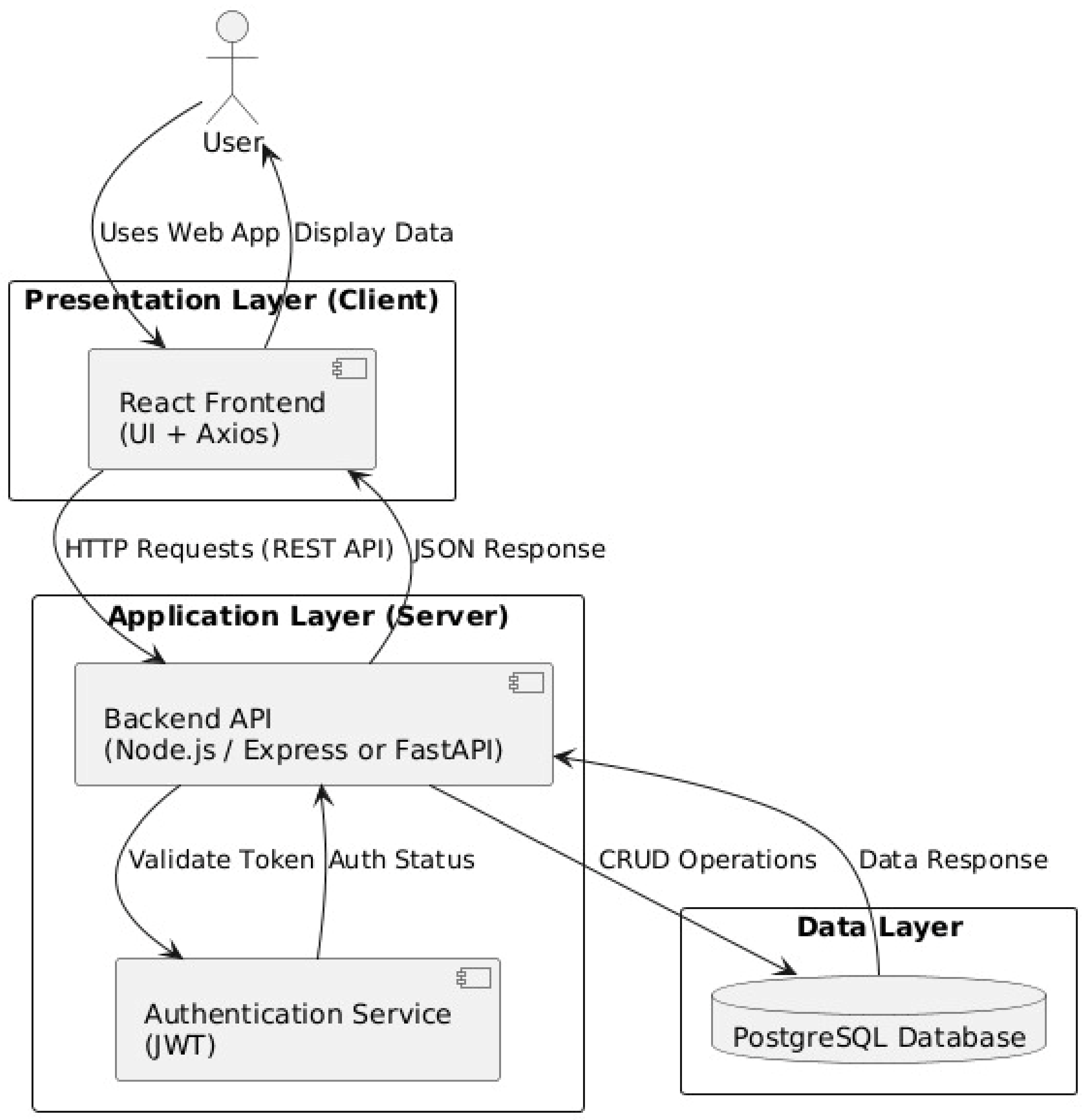
- **Frontend:** React, Axios
- **Backend:** Node.js (Express) / FastAPI
- **Database:** PostgreSQL
- **Authentication:** JWT
- **DevOps:** Docker, Docker Compose
- **CI/CD:** GitHub Actions
- **Database Management:** Adminer



2.1 Application Design

The application design focuses on usability and clarity. The interface is designed with minimal complexity so users can easily navigate through legal categories and access information.

3-Tier Client-Server Architecture - Legal Case Management System


User
Uses Web App
Display Data
Presentation Layer (Client)
React Frontend (UI + Axios)
HTTP Requests (REST API)
JSON Response
Application Layer (Server)
Backend API (Node.js / Express or FastAPI)
Validate Token
Auth Status
Authentication Service (JWT)
CRUD Operations
Data Response
Data Layer
PostgreSQL Database

2.2 Process Flow

The application design focuses on usability and clarity. The interface is designed with minimal complexity so users can easily navigate through legal categories and access information.
Responsive design techniques are used to ensure compatibility across different devices such as desktops, tablets, and mobile phones.

- User opens the web application
- Home page is displayed
- User selects a legal category

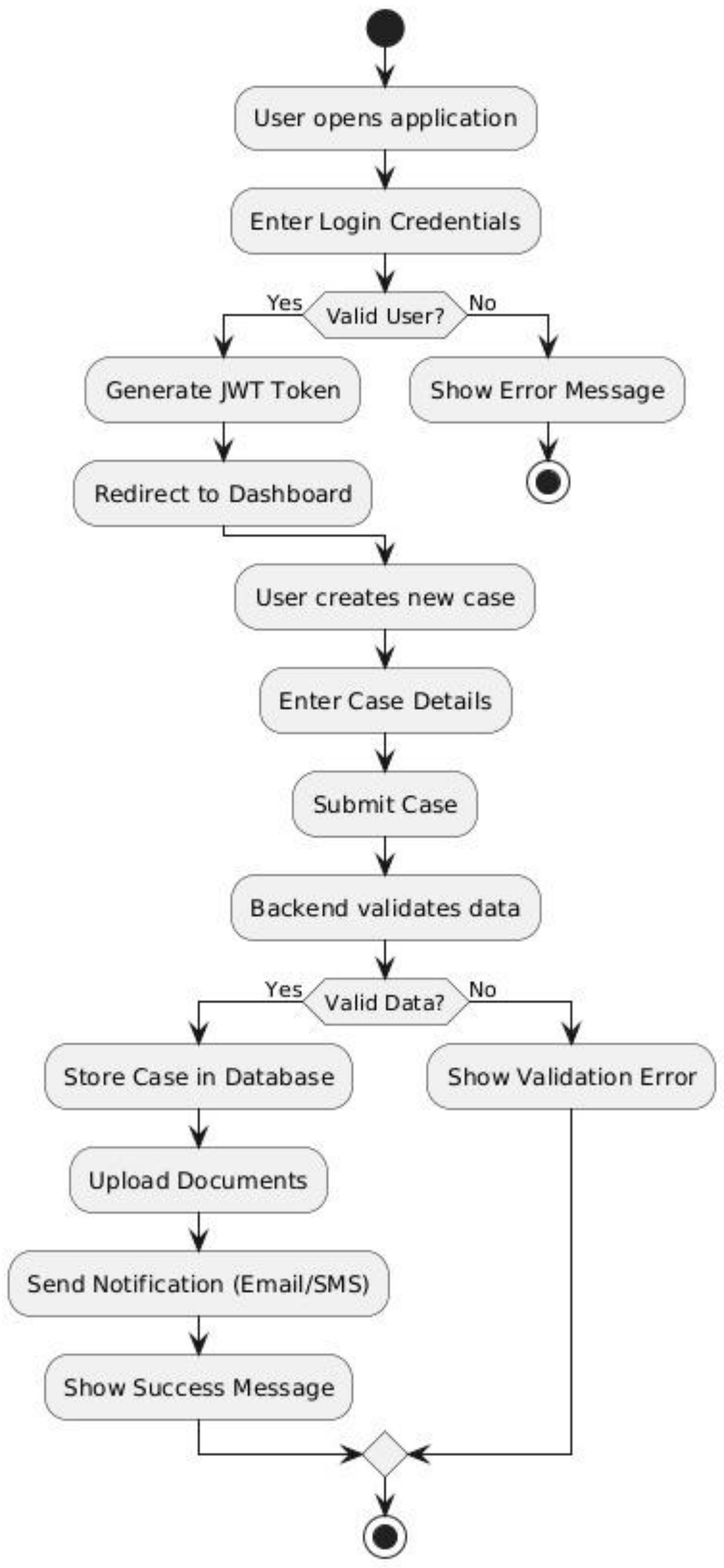


Fig 1. Activity Diagram

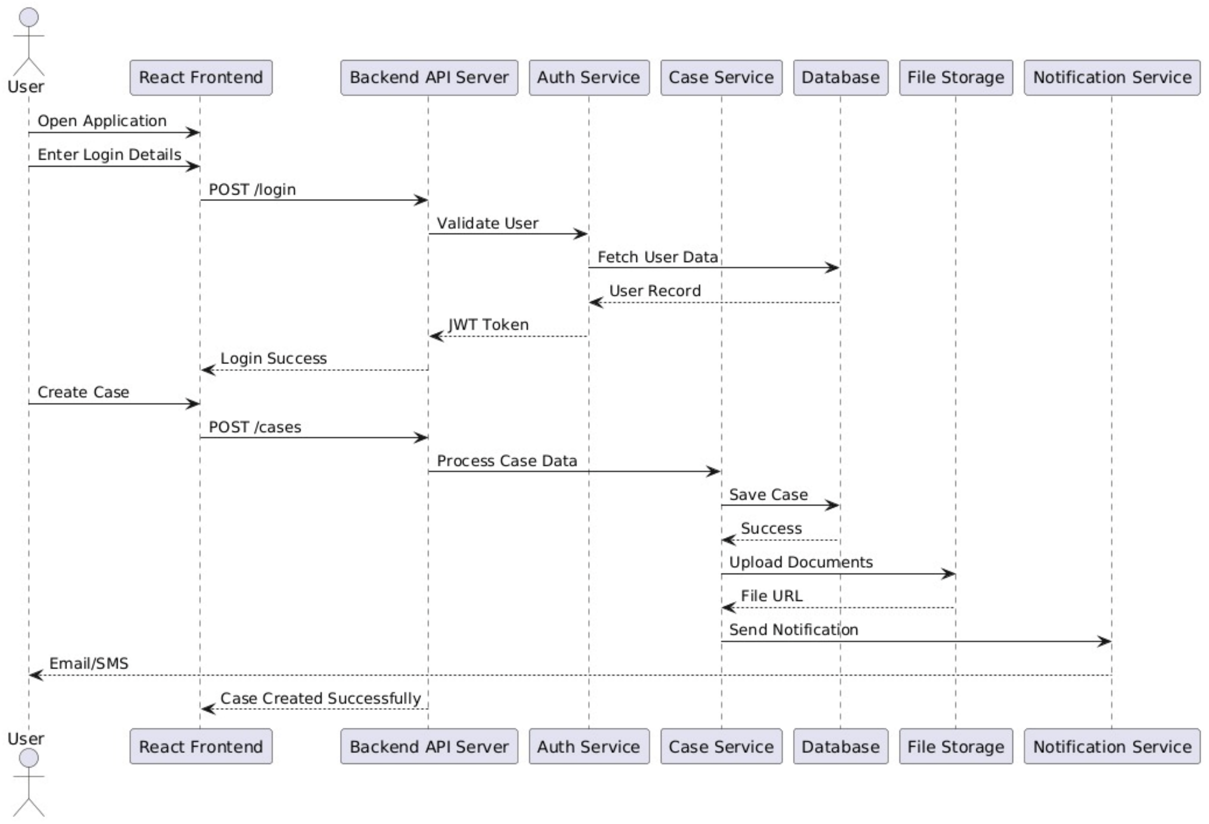


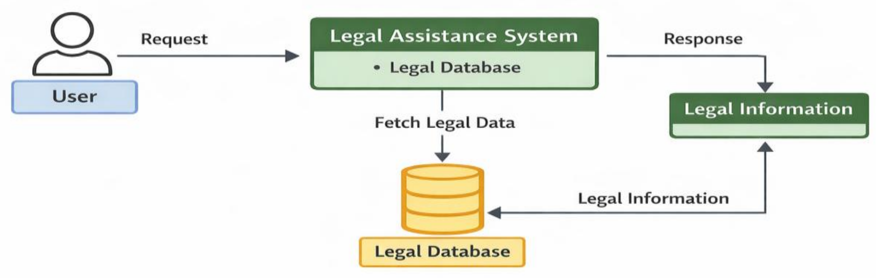
Fig 2. Sequence Diagram

2.3 Information Flow

Information flow describes how data moves within the system. User requests are processed by the application, and relevant legal information is retrieved and displayed.

The system does not allow users to directly manipulate data, ensuring content integrity.

- User requests information through the browser
- Application processes the request
- Relevant legal content is fetched
- Information is displayed on the user interface

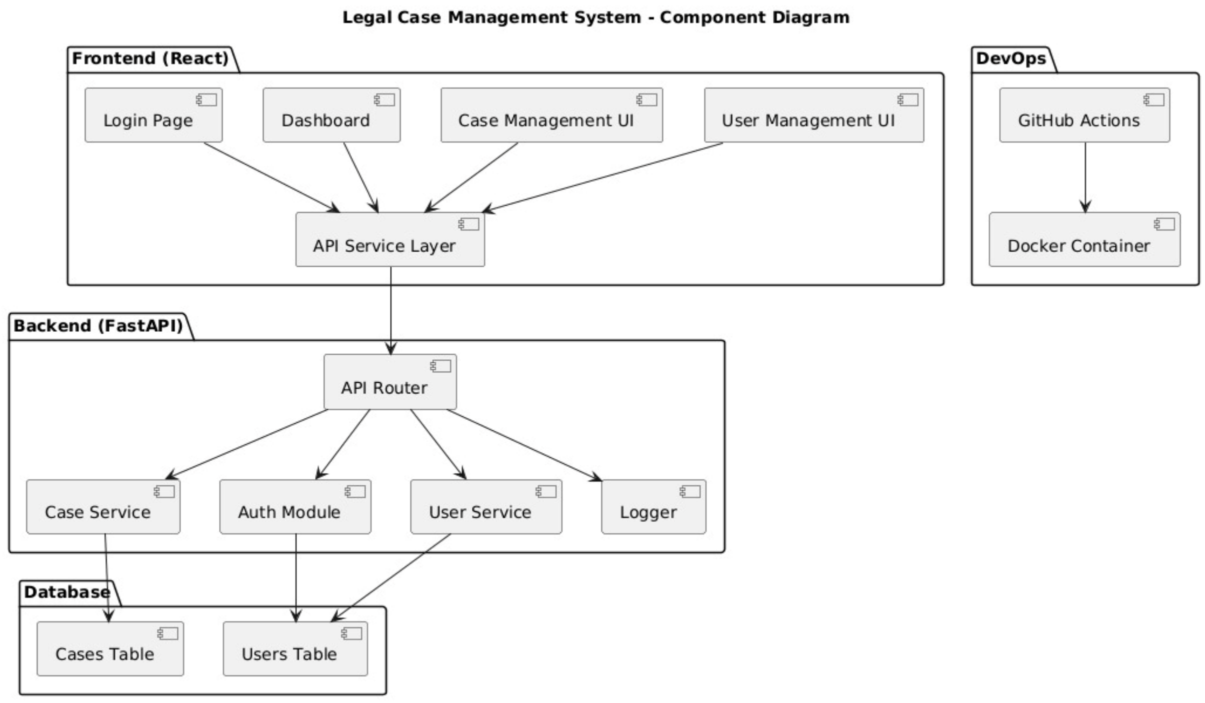


Information Flow Diagram

2.4 Components Design

In the Legal Case Management System, component design defines how the application is divided into independent modules such as frontend, backend, authentication, and database. Each component is responsible for a specific functionality and interacts with other components through APIs to ensure a scalable and maintainable system.

- **Modularity** – System is divided into frontend, backend, and database components
- **Loose Coupling** – Components communicate through APIs and are independent
- **High Cohesion** – Each component performs a specific task
- **Scalability** – Components can be scaled independently



2.5 Key Design Consideration

- Simple and clean UI
- Easy navigation
- Responsive design for mobile and desktop
- Scalability for future enhancements

2.6 API Catalogue

Currently, the application does not use external APIs.
Future versions can integrate:

- Legal databases
- Chatbot APIs
- Authentication APIs

3. Data Design

The Legal Case Management System currently uses a simple and organized data structure to manage legal information. At present, the system may use static or basic stored data, but it is designed in a way that can be easily extended to a database-driven approach in the future. Data consistency, clarity, and maintainability are ensured throughout the application.

3.1 Data Model

- Legal categories (e.g., case types, case status)
- Case details (title, description, status)
- User information (name, email, role)
- Document details related to cases

3.2 Data Access Mechanism

- Data is accessed through backend APIs
- Frontend fetches data using HTTP requests (Axios)
- Backend interacts with the database for CRUD operations
- Can be extended to more advanced data handling techniques

3.3 Data Retention Policies

- Basic user data is stored securely in the database
- Sensitive information can be protected using authentication (JWT)
- Data is maintained for application functionality
- Future enhancements can include encryption and backup policies

3.4 Data Migration

- Data migration is used to move or update data structure in the system
- Initial setup can include creating database tables and sample data
- Migration scripts can be used to modify schema (add/remove fields)
- Tools like ORM (Prisma / SQL Alchemy) can manage migrations
- Ensures data consistency during updates or deployment

4. Interfaces

Interfaces define how different parts of the system interact with each other and with external users. In this project, interfaces ensure smooth communication between the frontend, backend, database, and users.

Types of Interfaces

1. User Interface (UI)

- Developed using React for interactive web pages
- Provides screens like Login, Dashboard, Case Management
- Allows users to create, view, update, and delete cases
- Displays data dynamically based on API responses

---

2. API Interface (Frontend–Backend Communication)

- Uses REST APIs for communication
- Frontend sends HTTP requests (GET, POST, PUT, DELETE)
- Backend processes requests and returns JSON responses
- Axios is used to handle API calls in frontend

---

3. Backend–Database Interface

- Backend interacts with PostgreSQL database
- Performs CRUD operations (Create, Read, Update, Delete)
- Uses queries or ORM (Prisma / SQL Alchemy)
- Ensures data validation and consistency

---

4. Authentication Interface

- Handles user login and registration
- Uses JWT (JSON Web Token) for secure access
- Protects private routes and APIs
- Ensures only authorized users can access data

---

5. External / Admin Interface

- Adminer or similar tools used for database management
- Allows viewing and managing database tables
- Helps in debugging and data monitoring

5. State and Session Management

State Management

- State is managed in the **frontend (React)** to handle UI data
- Stores temporary data like user details, case data, and form inputs
- Updates UI dynamically without reloading the page
- Can be managed using React state or Context API
- Helps in maintaining application flow and responsiveness

Session Management

- Session is managed using **JWT (JSON Web Token)**
- After login, a token is generated and sent to the client
- Token is stored in browser (local Storage / cookies)
- Each API request includes the token for authentication
- Backend verifies token to allow or deny access

6. Caching

Caching is used in the Legal Case Management System to store frequently accessed data temporarily, improving performance and reducing load on the backend and database. It helps in faster response time and better user experience.

Caching in the Project

- Frequently used data (e.g., case lists, user details) can be cached
- Reduces repeated database queries
- Improves API response speed
- Enhances overall system performance

Types of Caching Used

- **Client-Side Caching**
    - Browser stores API responses temporarily
    - Reduces repeated requests to the server
- **Server-Side Caching (Future Scope)**
    - Tools like Redis can be used
    - Stores frequently accessed data in memory
- **API Caching**
    - Responses can be cached for specific time duration
    - Improves efficiency for repeated requests

7. Non-Functional Requirements

- **Performance** – Fast response time and efficient API processing
- **Scalability** – Can handle increasing users and data
- **Security** – JWT authentication and data protection
- **Reliability** – Stable system with proper error handling
- **Usability** – Simple and user-friendly interface
- **Maintainability** – Easy to update and debug code
- **Availability** – System accessible with minimal downtime

7.1 Security Aspects

- **Authentication** – Secure login using JWT tokens
- **Authorization** – Role-based access (admin/user)
- **Data Protection** – Sensitive data stored securely in database
- **Input Validation** – Prevents invalid or malicious inputs
- **API Security** – Protected routes with token verification
- **Password Security** – Passwords stored in encrypted/hashed form

7.2 Performance Aspects

- **Fast Response Time** – APIs should return data quickly
- **Efficient Database Queries** – Optimized queries to reduce load

- **Caching** – Store frequently accessed data to improve speed
- **Scalability** – System can handle multiple users simultaneously
- **Load Handling** – Supports concurrent requests efficiently
- **Frontend Optimization** – Reduce unnecessary re-renders
- **Asynchronous Processing** – Use async operations for better performance
- **Resource Utilization** – Efficient use of CPU and memory

8. References

- React Documentation – https://react.dev
- Node.js Documentation – https://nodejs.org/en/docs
- Express.js Documentation – https://expressjs.com
- Fast API Documentation – https://fastapi.tiangolo.com
- PostgreSQL Documentation – https://www.postgresql.org/docs
- Docker Documentation – https://docs.docker.com
- GitHub Actions Documentation – https://docs.github.com/en/actions