System Architecture

for

IKIGAI 2.O

Prepared by Salman Khan, Maryum Tanvir, Mavra Mehak

FAST - National University of Computer & Emerging Sciences

April 29th,2025

Architecture Style

The Ikigai/I-CARE Clinic Appointment Management System is built using the **MERN** stack (MongoDB, Express.js, React.js, Node.js) and follows the **Model-View-Controller** (MVC) architectural style.

- **Model:** The data layer is handled using MongoDB, where all hospital management-related data (e.g., doctor profiles, patient records, appointments) are stored. The models define the structure of the database collections and are used to interact with the database.
- **View:** The frontend is developed using React.js, which acts as the View in the MVC pattern. React is responsible for presenting data to users and providing a dynamic, responsive user interface. It communicates with the backend via APIs.
- **Controller**: The controllers are implemented in Express.js within the Node.js backend. Controllers handle the business logic, processing incoming HTTP requests, interacting with the models, and sending appropriate responses back to the client.

By using MVC, I-CARE separates concerns clearly, making the system more organized, easier to maintain, scalable, and testable. Each part (Model, View, Controller) is independently developed and maintained, improving collaboration and future extension of the application.

1) Deployment Diagram:

Nodes and Artifacts

Client Device

- **Description**: End user's device (desktop, tablet, mobile).
- **Artifact**: **Browser (View - MVC)**
    - Runs React frontend
    - Built with Vite (vite for dev, vite build for prod).
    - Features:

        - **Structure**: public/index.html (entry), src (pages: Dashboard.jsx, Appointments.jsx, Signup.jsx; components with antd, lucide-react icons; context: AuthContext.jsx, ThemeContext.jsx).
        - **Styling**: Tailwind CSS
        - **Data**: axios for API calls
        - **Routing**: react-router-dom for navigation.
        - **Date Handling**: dayjs for dates in Appointments.jsx, Shifts.jsx.
    - Communicates with Application Server via HTTP/HTTPS (enabled by cors).

Application Server

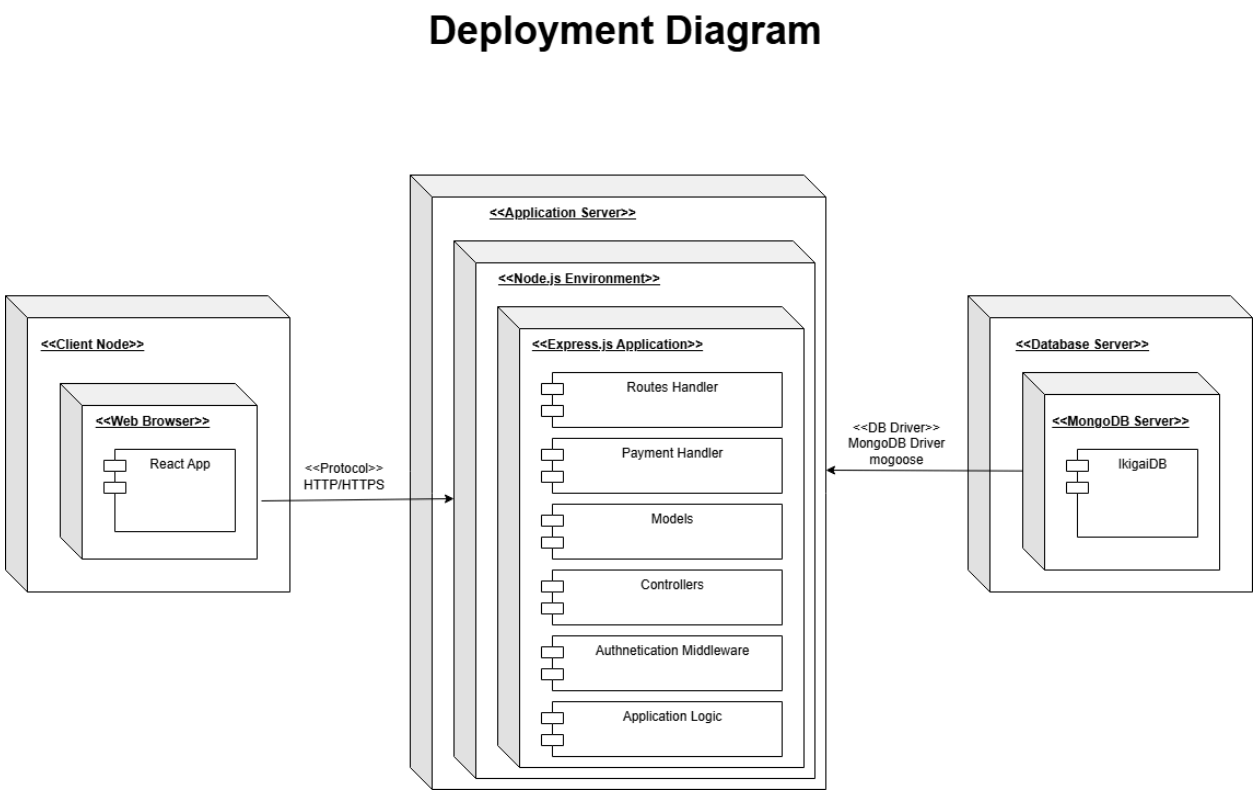
- **Description**: Cloud-based or on-premises server hosting Node.js/Express.js backend.
- **Artifact**: **Node.js Runtime (Controller - MVC)**
    - Runs backend.
    - Features:
        - **Entry**: server.js (express, cors, morgan for logging).
        - **Routes**: API endpoints (admin.routes.js, appointment.routes.js, auth.routes.js).
        - **Controllers**: Business logic (admin_controller.js, appointment_controller.js).
        - **Middlewares**: auth_middleware.js (jsonwebtoken, bcryptjs), error_middleware.js, multer_middleware.js (multer, path, uuid).
        - **Config**: dotenv for .env (e.g., DB URI, JWT secret).
    - Connects to Database Server via mongoose.

Database Server

- **Description**: Server hosting MongoDB.
- **Artifact**: **MongoDB Server (Model - MVC)**
    - Stores I-CARE database with collections (AdminProfile.js, Appointment.js, DoctorRequest.js, Feedback.js, PatientProfile.js, Payment.js, Shift.js, User.js).
    - Accessed by Application Server via mongoose for CRUD operations.

Connections

- **Client Device (Browser) ↔ Application Server**: HTTP/HTTPS requests (enabled by cors).
- **Application Server ↔ Database Server**: MongoDB protocol via mongoose.



2) Package Diagram

I-CARE (Top-Level Package)

The main package encompassing the entire application.

Frontend (Package)

Encapsulates client-side code, built with React.js and Vite.

- **Sub-Packages**:
    - **src**: Core application code.

    - **pages**: React components for views (e.g., Dashboard.jsx, Appointments.jsx, Signup.jsx) for admin, auth, doctor, patient.
    - **components**: Reusable UI components (e.g., generic/AppointmentCard.jsx), using antd for UI and lucide-react for icons.
    - **context**: Global state management (e.g., AuthContext.jsx, ThemeContext.jsx).
    - **assets**: Static assets (e.g., images/logo.png, styles/global.css).
    - **hooks**: Custom hooks (e.g., useAuth.js) for authentication logic.
- **Configuration**:
    - vite.config.js
    - tailwind.config.js
    - eslint.config.js
- **Dependencies**:
    - react-router-dom for routing.
    - axios for API calls.
    - dayjs for date handling.
- **Dependency Arrow**: src → backend for API access.

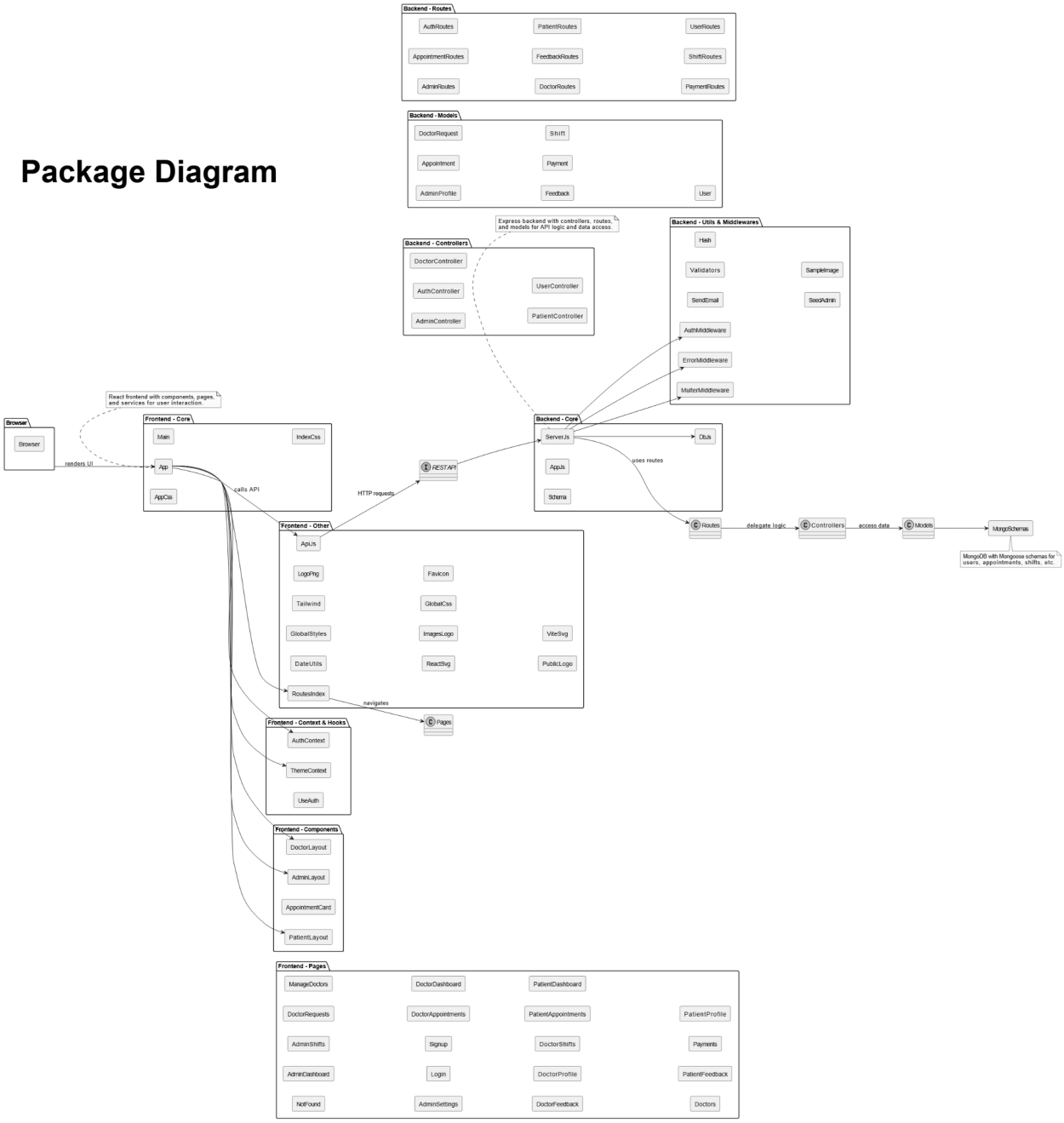
Backend (Package)

Encapsulates server-side code built with Node.js and Express.js.

- **Sub-Packages**:
    - **routes**: API endpoints (e.g., admin.routes.js, appointment.routes.js), using express.
    - **controllers**: Business logic (e.g., admin_controller.js, appointment_controller.js).
    - **models**: MongoDB schemas (e.g., AdminProfile.js, Appointment.js), using mongoose.
    - **middlewares**:
        - auth_middleware.js (with jsonwebtoken, bcryptjs).
        - error_middleware.js
- **Configuration**:
    - dotenv for environment variables (.env).
- **Dependencies**:
    - cors for cross-origin requests.
    - nodemon for development.

- **Dependency Arrows**:
    - routes → models for database access.
    - controllers → middlewares for request processing.

Package Diagram



3) Component Diagram

Frontend Component (React.js)

- **Location**: frontend/src
- **Sub-Components**:
    - **Admin**: Dashboard.jsx, DoctorRequests.jsx, ManageDoctors.jsx, Settings.jsx (using antd for UI, lucide-react for icons).
    - **Doctor**: DoctorDashboard.jsx, Appointments.jsx, Feedback.jsx, Profile.jsx, Shifts.jsx (using dayjs for date handling).
    - **Patient**: Dashboard.jsx, Doctors.jsx, Appointments.jsx, Feedback.jsx, Payments.jsx, Profile.jsx.
    - **Auth**: Signup.jsx, Login.jsx (with AuthContext.jsx, useAuth.js).
- **Routing & Data**:
    - react-router-dom for navigation (index.jsx).
    - axios for API requests.
- **Styling**: Tailwind CSS (styles/global.css) for responsive design.
- **Interfaces**: HTTP interface for API calls to Backend Component (e.g., GET /doctors, POST /appointments) via axios.

Backend Component (Node.js/Express.js)

- **Location**: backend
- **Sub-Components**:
    - **Controller**: admin_controller.js, appointment_controller.js, auth_controller.js (business logic with express).
    - **Middleware**:
        - auth_middleware.js (jsonwebtoken for JWT, bcryptjs for password hashing).
        - error_middleware.js.
        - multer_middleware.js (multer for file uploads, path, uuid).
    - **Model**: Mongoose schemas (AdminProfile.js, Appointment.js, etc.) for MongoDB via mongoose.
- **Setup**:

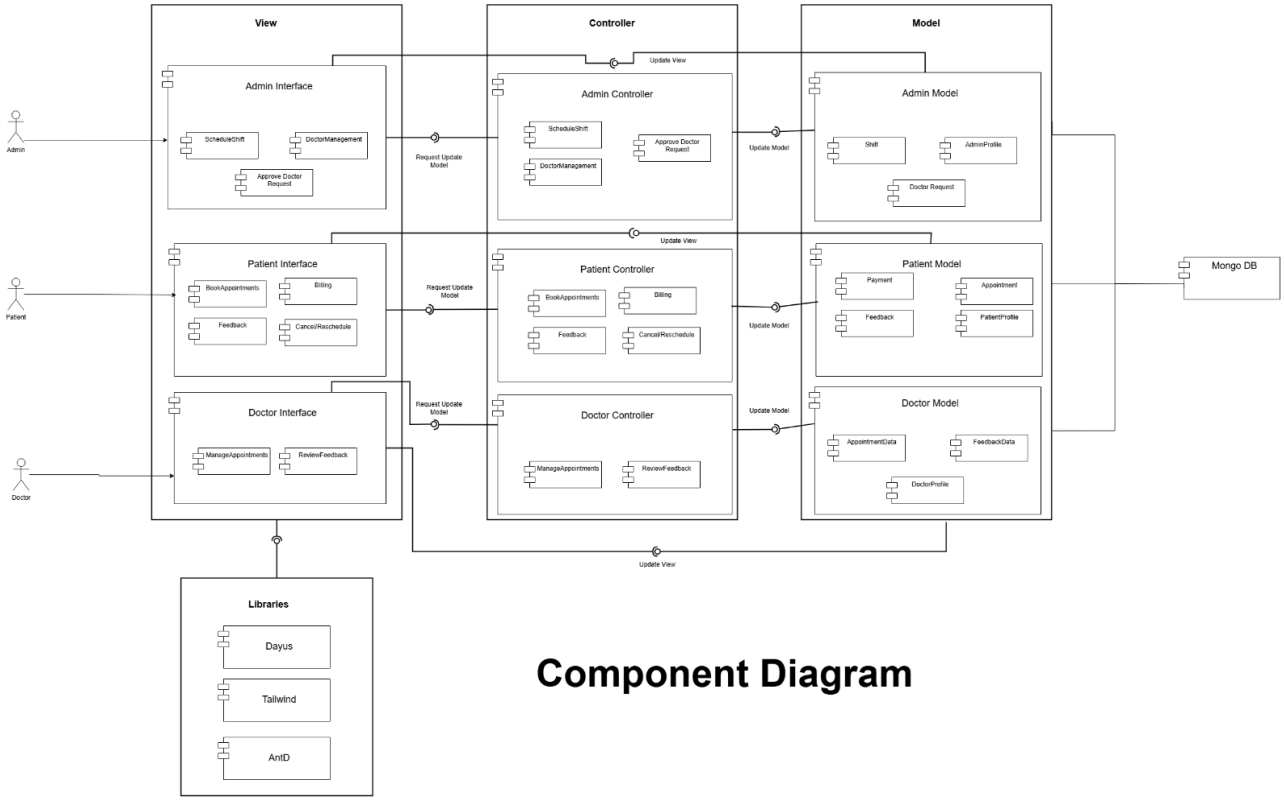
    - cors for cross-origin requests.
- **Interfaces**:
    - API endpoints (/admin, /appointments, etc.) via routes for Frontend Component.
    - MongoDB connection via mongoose.

Database Component (MongoDB)

- **Description**: Stores collections (admins, doctors, patients, appointments, feedback, payments, shifts, users).
- **Interfaces**: MongoDB connection interface via mongoose for Backend Component.

Interactions

- **Frontend → Backend**: HTTP requests (via axios) to API endpoints, enabled by cors.
- **Backend (Controller) → Middleware**: Request processing (e.g., authentication with jsonwebtoken, file uploads with multer).
- **Backend (Controller) → Model**: Database access via mongoose to interact with Database Component.



Component Diagram