# **Clinic Management System - Documentation**

## **1. Project Overview**

This document provides a comprehensive overview of the Clinic Management System, a full-stack web application designed to streamline clinic operations. The system provides a robust set of features for managing patients, doctors, appointments, medical records, and more. It is built with a modern technology stack, featuring a React frontend and a FastAPI backend, ensuring a scalable, maintainable, and efficient solution.

### **1.1. Core Features**

* **Role-Based Access Control:** Differentiated dashboards and permissions for Admins, Doctors, Staff, and Patients.
* **User Management:** Secure user registration, login, profile management, and password recovery.
* **Appointment Scheduling:** Patients can book appointments, and doctors/staff can manage schedules.
* **Electronic Medical Records (EMR):** Doctors can create, view, and manage patient EMRs, including medical history and prescriptions.
* **Consultation Interface:** A dedicated interface for doctors to conduct consultations.
* **Billing & Invoicing:** Staff can manage billing for services.
* **Reporting & Analytics:** Admins can view system-wide reports and analytics.
* **Integrated Chatbot:** A helpful chatbot to assist users.

### **1.2. Technology Stack**

* **Backend:**
  + **Framework:** FastAPI
  + **Database:** SQLAlchemy ORM (compatible with PostgreSQL, SQLite, etc.)
  + **Data Validation:** Pydantic
  + **Authentication:** JWT (JSON Web Tokens)
* **Frontend:**
  + **Framework:** React (with Create React App)
  + **Language:** TypeScript
  + **Routing:** React Router
  + **Styling:** CSS Modules, standard CSS

## **2. System Architecture**

The application follows a classic client-server architecture.

* **Frontend (Client):** A single-page application (SPA) built with React. It is responsible for rendering the user interface and interacting with the backend API. It does not contain any business logic related to data persistence.
* **Backend (Server):** A RESTful API built with FastAPI. It handles all business logic, database interactions, and authentication. It exposes a series of endpoints that the frontend consumes.
* **Database:** A relational database that persists all application data. The schema is managed by SQLAlchemy models.

+-----------------+ HTTP (REST API) +-----------------+ SQL +-----------------+  
| | <----------------------> | | <------------>| |  
| React Client | | FastAPI Server | | Database |  
| (Browser) | | (Business Logic) | | (PostgreSQL) |  
| | ----------------------> | | | |  
+-----------------+ +-----------------+ +-----------------+

## **3. Getting Started: Setup & Installation**

Follow these steps to set up and run the project on a local machine.

### **3.1. Prerequisites**

* Python 3.8+ and pip
* Node.js v14+ and npm
* A running PostgreSQL server (or you can modify the code to use SQLite)

### **3.2. Backend Setup**

1. **Navigate to the backend directory:**  
   cd backend
2. **Create and activate a virtual environment:**  
   python -m venv venv  
   source venv/bin/activate # On Windows, use `venv\Scripts\activate`
3. **Install Python dependencies:**  
   pip install -r ../requirements.txt
4. **Configure the Database:**
   * Open backend/app/config.py.
   * Update the SQLALCHEMY\_DATABASE\_URL with your database connection string. For PostgreSQL, it looks like this:  
     "postgresql://user:password@host:port/database\_name"
5. **Run Database Migrations (Creating Tables):**
   * The current setup creates tables automatically on startup via database.py. Ensure your database is running before starting the server.
6. **Create an Initial Admin User (Optional but Recommended):**
   * Run the helper script from the backend/app directory:

python create\_initial\_admin.py

* + This will create a default admin user for you to log in with.

1. **Run the Backend Server:**
   * From the backend/app directory:

uvicorn main:app --reload

* + The API will be available at http://127.0.0.1:8000. You can access the interactive API documentation (Swagger UI) at http://127.0.0.1:8000/docs.

### **3.3. Frontend Setup**

1. **Navigate to the frontend directory:**  
   cd frontend
2. **Install Node.js dependencies:**  
   npm install
3. **Run the Frontend Development Server:**  
   npm start
4. **Access the Application:**
   * Open your web browser and navigate to http://localhost:3000. The React application will automatically connect to the backend server running on port 8000.

## **4. Backend API Endpoints**

The backend is organized into several routers, each handling a specific domain. All API endpoints are prefixed with /api.

### **4.1. Authentication (/api/auth)**

* **POST /token**: Login. Takes a form with username and password. Returns an access\_token and token\_type.
* **POST /register**: Register a new user. Expects a UserCreate schema in the request body. Returns the newly created User object.
* **POST /request-password-reset/{email}**: Initiates the password reset process for a given email.
* **POST /reset-password/**: Resets the password. Requires a token and new\_password in the request body.

### **4.2. Users (/api/users)**

* **GET /me**: Get the profile of the currently logged-in user. Requires authentication.
* **PUT /me**: Update the profile of the currently logged-in user. Requires authentication and a UserUpdate schema in the body.
* **GET /**: Get a list of all users. Requires admin privileges.
* **GET /{user\_id}**: Get details of a specific user by ID. Requires admin privileges.
* **DELETE /{user\_id}**: Delete a user by ID. Requires admin privileges.

### **4.3. Patients (/api/patients)**

* **GET /**: Get a list of all patients.
* **GET /search/**: Search for patients by name (q).
* **GET /{patient\_id}**: Get details of a specific patient by ID.

### **4.4. Doctors (/api/doctors)**

* **GET /**: Get a list of all doctors.
* **GET /{doctor\_id}/schedule**: Get the schedule for a specific doctor.

### **4.5. Appointments (/api/appointments)**

* **POST /**: Create a new appointment. Requires authentication and an AppointmentCreate schema.
* **GET /**: Get a list of all appointments.
* **GET /user/{user\_id}**: Get all appointments for a specific user.
* **GET /doctor/{doctor\_id}**: Get all appointments for a specific doctor.
* **PUT /{appointment\_id}**: Update an appointment (e.g., change status). Requires a status in the body.
* **DELETE /{appointment\_id}**: Delete an appointment.

### **4.6. Reports / EMR (/api/reports)**

* **POST /**: Create a new medical report (EMR). Requires authentication and a ReportCreate schema.
* **GET /patient/{patient\_id}**: Get all reports for a specific patient.
* **GET /{report\_id}**: Get a specific report by its ID.

### **4.7. Chat (/api/chat)**

* **POST /**: Send a message to the chatbot. Expects a Message schema (content: str). Returns a chatbot response.

## **5. Frontend Structure**

The frontend code is organized into a logical and reusable component structure.

### **5.1. Main Directories & Files**

* **src/components**: Contains all reusable React components, which form the building blocks of the UI.
  + **admin/**: Components specific to the Admin dashboard (User Management, Billing, etc.).
  + **appointment/**: Components for booking, viewing, and managing appointments.
  + **dashboards/**: High-level dashboard components for each user role (Admin, Doctor, Patient, Staff).
  + **layout/**: Structural components like Sidebar and BaseDashboard that define the overall page layout.
  + **medical/**: Components for displaying medical history, prescriptions, and reports.
  + **auth/**: Login, Register, and Password Reset form components (these appear to be in the main components dir, like LoginPage.tsx).
* **src/pages**: Components that represent entire pages, typically composed of multiple smaller components (e.g., Dashboard.tsx, Profile.tsx).
* **src/styles**: Global CSS files and style definitions.
* **src/types**: TypeScript type and interface definitions.
* **App.tsx**: The root component. It sets up the react-router-dom to handle all client-side routing.
* **index.tsx**: The entry point of the application, which renders the App component into the DOM.

### **5.2. Component High-level view**

* **App.tsx**: This is the core of the frontend. It uses BrowserRouter to define all the application routes, such as /login, /register, /dashboard, /profile, etc. It conditionally renders different pages based on the current URL.
* **BaseDashboard.tsx**: A crucial layout component. It provides the common structure for all authenticated pages, including the Sidebar and a main content area where child components (the actual page content) are rendered.
* **Sidebar.tsx**: The navigation component, which displays different links based on the logged-in user's role (userType).
* **Role-Based Dashboards (AdminDashboard.tsx, DoctorDashboard.tsx, etc.)**: These components are the main landing pages for each user after they log in. They compose various smaller components to display relevant information and actions. For example, DoctorDashboard shows upcoming appointments, while AdminDashboard shows user statistics and system-wide reports.
* **Authentication Flow**:
  1. The user starts at /login (LoginPage.tsx).
  2. Upon successful login, an auth token is saved to local storage, and the user is redirected to /dashboard.
  3. The DashboardWrapper.tsx component fetches the user's profile using the saved token.
  4. Based on the user.role from the profile, it renders the appropriate dashboard (e.g., <AdminDashboard />).
  5. All subsequent API requests include the auth token in the header to access protected routes.

## **6. Database Schema**

The database schema is defined in backend/app/models.py using SQLAlchemy. The primary models are:

* **User**
  + id (PK)
  + email (Unique)
  + username (Unique)
  + full\_name
  + hashed\_password
  + role (e.g., 'admin', 'doctor', 'patient', 'staff')
  + is\_active
  + *Relationships*: Has one Patient or Doctor profile. Has many Appointments.
* **Patient**
  + id (PK)
  + user\_id (FK to User)
  + date\_of\_birth
  + address
  + phone\_number
  + *Relationships*: Belongs to one User. Has many Appointments and Reports.
* **Doctor**
  + id (PK)
  + user\_id (FK to User)
  + specialty
  + license\_number
  + *Relationships*: Belongs to one User. Has many Appointments.
* **Appointment**
  + id (PK)
  + patient\_id (FK to Patient)
  + doctor\_id (FK to Doctor)
  + appointment\_date
  + reason
  + status (e.g., 'scheduled', 'completed', 'canceled')
  + *Relationships*: Belongs to one Patient and one Doctor.
* **Report** (EMR)
  + id (PK)
  + patient\_id (FK to Patient)
  + doctor\_id (FK to Doctor)
  + report\_date
  + diagnosis
  + treatment
  + notes
  + *Relationships*: Belongs to one Patient and one Doctor.