## **🔥 Deep ER Model for Online Healthcare System 🔥**

### **1️⃣ Patient (Strong Entity)**

* **Patient\_ID** (PK)
* First\_Name
* Last\_Name
* Date\_of\_Birth
* Gender
* Blood\_Type
* Contact\_Number
* Email
* Address
* Emergency\_Contact
* Insurance\_Details
* Username (For Login)
* Password (For Login)
* **Relationships:**
  + One Patient → Can Have Multiple **Appointments** (1:M)
  + One Patient → Can Upload Multiple **Reports** (1:M)
  + One Patient → Can Receive Multiple **Prescriptions** (1:M)

### **2️⃣ Doctor (Strong Entity)**

* **Doctor\_ID** (PK)
* First\_Name
* Last\_Name
* Specialization (Neurology, Cardiology, etc.)
* Experience\_Years
* Contact\_Number
* Email
* Consultation\_Fees
* Availability\_Status (Available/Not Available)
* Clinic\_Hospital\_Name
* Username (For Login)
* Password (For Login)
* **Relationships:**
  + One Doctor → Can Have Multiple **Appointments** (1:M)
  + One Doctor → Can Issue Multiple **Prescriptions** (1:M)

### **3️⃣ Appointment (Associative Entity)**

* **Appointment\_ID** (PK)
* **Patient\_ID** (FK)
* **Doctor\_ID** (FK)
* Date
* Time
* Status (Scheduled, Completed, Cancelled)
* Mode (Online / In-person)
* Payment\_Status (Paid / Unpaid)
* **Relationships:**
  + Links **Patient** and **Doctor**
  + One Appointment → Can Generate a **Medical Report** (1:1)

### **4️⃣ Medical\_Report (Weak Entity, Depends on Appointment)**

* **Report\_ID** (PK)
* **Appointment\_ID** (FK)
* **Patient\_ID** (FK)
* **Doctor\_ID** (FK)
* Diagnosis
* Symptoms
* Test\_Recommended
* Test\_Results
* Uploaded\_Report\_File
* Date
* **Relationships:**
  + One Medical Report → Is Associated with One **Appointment** (1:1)

### **5️⃣ Prescription (Weak Entity, Depends on Doctor and Patient)**

* **Prescription\_ID** (PK)
* **Patient\_ID** (FK)
* **Doctor\_ID** (FK)
* **Appointment\_ID** (FK)
* Medicine\_Name
* Dosage
* Frequency (e.g., Twice a Day)
* Instructions
* Refill (Yes/No)
* Date\_Issued
* **Relationships:**
  + One Prescription → Belongs to One **Appointment** (1:1)

### **6️⃣ Payment (Weak Entity, Linked to Appointment)**

* **Payment\_ID** (PK)
* **Appointment\_ID** (FK)
* Amount
* Payment\_Mode (Credit Card, UPI, Cash)
* Transaction\_ID
* Payment\_Status (Success / Failed)
* **Relationships:**
  + One Payment → Linked to One **Appointment** (1:1)

### **7️⃣ Admin (Manages System, Strong Entity)**

* **Admin\_ID** (PK)
* Name
* Contact
* Email
* Username (For Login)
* Password (For Login)
* **Relationships:**
  + Can Add/Remove **Doctors**
  + Can View **Patient Reports**

### **8️⃣ Feedback & Rating (Weak Entity, Linked to Patient & Doctor)**

* **Feedback\_ID** (PK)
* **Patient\_ID** (FK)
* **Doctor\_ID** (FK)
* Rating (1 to 5)
* Comments
* Date
* **Relationships:**
  + One Patient → Can Give Multiple Feedbacks (1:M)

## **🔗 Deep Relationships 🔗**

1. **Patient & Doctor (M:M)** → Through **Appointment**
2. **Patient & Medical Report (1:M)**
3. **Doctor & Prescription (1:M)**
4. **Appointment & Payment (1:1)**
5. **Appointment & Medical Report (1:1)**
6. **Patient & Feedback (1:M)**
7. **Doctor & Feedback (1:M)**

Table Patient {

Patient\_ID int [primary key]

First\_Name varchar

Last\_Name varchar

Date\_of\_Birth date

Gender varchar

Blood\_Type varchar

Contact\_Number varchar

Email varchar

Address varchar

Emergency\_Contact varchar

Insurance\_Details varchar

Username varchar

Password varchar

}

Table Doctor {

Doctor\_ID int [primary key]

First\_Name varchar

Last\_Name varchar

Specialization varchar

Experience\_Years int

Contact\_Number varchar

Email varchar

Consultation\_Fees decimal

Availability\_Status varchar

Clinic\_Hospital\_Name varchar

Username varchar

Password varchar

}

Table Appointment {

Appointment\_ID int [primary key]

Patient\_ID int [ref: > Patient.Patient\_ID]

Doctor\_ID int [ref: > Doctor.Doctor\_ID]

Date date

Time time

Status varchar

Mode varchar

Payment\_Status varchar

}

Table Medical\_Report {

Report\_ID int [primary key]

Appointment\_ID int [ref: > Appointment.Appointment\_ID]

Patient\_ID int [ref: > Patient.Patient\_ID]

Doctor\_ID int [ref: > Doctor.Doctor\_ID]

Diagnosis varchar

Symptoms text

Test\_Recommended text

Test\_Results text

Uploaded\_Report\_File varchar

Date date

}

Table Prescription {

Prescription\_ID int [primary key]

Patient\_ID int [ref: > Patient.Patient\_ID]

Doctor\_ID int [ref: > Doctor.Doctor\_ID]

Appointment\_ID int [ref: > Appointment.Appointment\_ID]

Medicine\_Name varchar

Dosage varchar

Frequency varchar

Instructions text

Refill boolean

Date\_Issued date

}

Table Payment {

Payment\_ID int [primary key]

Appointment\_ID int [ref: > Appointment.Appointment\_ID]

Amount decimal

Payment\_Mode varchar

Transaction\_ID varchar

Payment\_Status varchar

}

Table Admin {

Admin\_ID int [primary key]

Name varchar

Contact varchar

Email varchar

Username varchar

Password varchar

}

Table Feedback {

Feedback\_ID int [primary key]

Patient\_ID int [ref: > Patient.Patient\_ID]

Doctor\_ID int [ref: > Doctor.Doctor\_ID]

Rating int

Comments text

Date date

}

# **Project Overview**

The **Online Healthcare System** is a full-stack web application designed to streamline patient-doctor interactions, appointment scheduling, and medical record management. This project primarily focuses on backend development using **MongoDB, Express.js, Node.js**, and frontend integration using **React.js**. The system ensures seamless healthcare management by enabling secure storage and retrieval of patient records, efficient doctor-patient communication, and an advanced appointment booking system.

## **💡 Features**

### **🔹 User Management**

* **Patients** can register, log in, and manage their profiles.
* **Doctors** can register, log in, and update availability.
* **Admins** can manage users, doctors, and system settings.
* **Two-Factor Authentication (2FA)** for secure login.

### **🔹 Appointment System**

* Patients can search for doctors based on specialization.
* Booking appointments with available doctors.
* Doctors can accept, reschedule, or cancel appointments.
* Email and SMS notifications for appointment confirmations.
* Automated reminders for upcoming appointments.

### **🔹 Medical Records & Prescriptions**

* Patients can upload and manage their medical reports.
* Doctors can prescribe medicines and generate reports.
* Secure encryption of medical history for privacy.
* Integration with **OCR** (Optical Character Recognition) to scan and digitize reports.

### **🔹 Payments & Billing**

* Integration of **Stripe/Razorpay/PayPal** for online payments.
* Payment status tracking and invoices.
* Automated billing for consultation and treatments.

### **🔹 Feedback & Reviews**

* Patients can rate doctors and provide feedback.
* Reviews are stored in MongoDB and analyzed for insights.
* AI-powered sentiment analysis to detect patient satisfaction levels.

### **🔹 Real-Time Communication**

* **Chat System:** Secure doctor-patient chat for quick consultations.
* **Video Consultation:** Real-time video calls with doctors.
* **Socket.io Integration:** Live updates on appointment status.

### **🔹 Security & Compliance**

* **JWT Authentication & Role-Based Access Control (RBAC).**
* **HIPAA Compliance** for secure handling of medical data.
* **Data Encryption** for storing patient information.
* **Multi-Factor Authentication (MFA)** for enhanced security.

## **🛠️ Tech Stack**

### **Backend**

* **Node.js** – Server-side logic.
* **Express.js** – API routes and middleware.
* **MongoDB** – NoSQL database for storing user and medical records.
* **Mongoose** – ODM for MongoDB.
* **JWT** – Authentication & Authorization.
* **Cloudinary/AWS S3** – For storing medical reports.

### **Frontend**

* **React.js** – User interface.
* **Redux** – State management.
* **TailwindCSS** – Styling.

### **Other Tools & Integrations**

* **Docker** – Containerized deployment.
* **Postman** – API testing.
* **Nodemailer** – Sending appointment emails.
* **Socket.io** – Real-time chat between patients and doctors.
* **GraphQL** – Alternative API option for data fetching.
* **AI Chatbot** – Provides instant medical guidance.

## **📂 Database Schema**

### **Collections & Structure**

#### **Patient**

{

"Patient\_ID": "ObjectId",

"First\_Name": "string",

"Last\_Name": "string",

"Date\_of\_Birth": "date",

"Gender": "string",

"Blood\_Type": "string",

"Contact\_Number": "string",

"Email": "string",

"Address": "string",

"Emergency\_Contact": "string",

"Insurance\_Details": "string",

"Username": "string",

"Password": "string"

}

#### **Doctor**

{

"Doctor\_ID": "ObjectId",

"First\_Name": "string",

"Last\_Name": "string",

"Specialization": "string",

"Experience\_Years": "number",

"Contact\_Number": "string",

"Email": "string",

"Consultation\_Fees": "number",

"Availability\_Status": "string",

"Clinic\_Hospital\_Name": "string",

"Username": "string",

"Password": "string"

}

### **Other Collections:**

* **Appointment** (Links Patients & Doctors)
* **Medical Reports**
* **Prescriptions**
* **Payments**
* **Feedback & Ratings**

## **🚀 Backend Development Roadmap**

### **1️⃣ Project Setup**

* Initialize Node.js project (npm init -y).
* Install dependencies (express, mongoose, dotenv, jsonwebtoken, bcrypt, etc.).
* Set up **MongoDB Atlas** and connect using Mongoose.
* Create .env file for secret configurations.

### **2️⃣ Database Modeling**

* Define **Mongoose Schemas** for all entities (Patient, Doctor, Appointment, etc.).
* Establish relationships using ObjectId references.
* Implement validation for user input.

### **3️⃣ Authentication & Authorization**

* Implement **JWT-based authentication**.
* Hash passwords using bcrypt.
* Role-based access control (Admin, Doctor, Patient).
* Implement OAuth login (Google, Facebook) for easy sign-up.

### **4️⃣ API Development**

* Create RESTful routes for **CRUD operations**.
* Develop controllers for handling logic.
* Use **Express.js** middleware for request validation.

### **5️⃣ Implementing Features**

* **Appointment Booking System**
* **Medical Report Upload & Storage**
* **Doctor Prescriptions Management**
* **Payment Gateway Integration**
* **Real-time Notifications using Socket.io**

### **6️⃣ Security Enhancements**

* Implement **rate limiting** to prevent abuse.
* Use **Helmet & CORS** for API security.
* Encrypt sensitive data before storing.

### **7️⃣ Testing & Deployment**

* Use **Postman** for API testing.
* Deploy backend using **Docker & Kubernetes**.
* Host on **AWS, Heroku, or DigitalOcean**.

## **⚙️ Installation & Setup**

### **1️⃣ Clone the Repository**

git clone https://github.com/your-repo/online-healthcare-system.git

cd online-healthcare-system

### **2️⃣ Install Dependencies**

npm install # Backend dependencies

cd frontend && npm install # Frontend dependencies

### **3️⃣ Set Up Environment Variables**

Create a .env file in the root directory with:

PORT=5000

MONGO\_URI=your-mongodb-uri

JWT\_SECRET=your-secret-key

STRIPE\_SECRET=your-stripe-key

CLOUDINARY\_URL=your-cloudinary-url

### **4️⃣ Start the Application**

npm run dev # Starts backend and frontend concurrently

## **🚀 Future Enhancements**

* Implement AI-based **doctor recommendations**.
* Add **video consultation** feature.
* Enable **blockchain-based medical record security**.
* Introduce **machine learning** for predictive diagnosis.
* Implement **IoT integration** for real-time health monitoring.

## **🤝 Contribution Guidelines**

* Fork the repository.
* Create a new branch (feature-xyz).
* Commit your changes.
* Open a Pull Request.

## **📜 License**

This project is licensed under the **MIT License**.

Let me know if you need any further improvements! 🚀