|  |  |  |  |
| --- | --- | --- | --- |
|  | BỘ GIÁO DỤC VÀ ĐÀO TẠO  **TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM** |  |  |

**ĐỒ ÁN CHUYÊN NGÀNH**

**TELEHEALTH SYSTEM**

**HỆ THỐNG ĐẶT LỊCH VÀ THEO DÕI KHÁM BỆNH TỪ XA**

Ngành: **Công nghệ thông tin**

Giảng viên hướng dẫn: **Thầy Đỗ Gia Bảo**

Sinh viên thực hiện**: Hồ Hoàng Nhân**

MSSV: **2310060131**

Lớp: **23TXTH02**

*TP. Hồ Chí Minh, ngày 26 tháng 08 năm 2025*

# **LỜI CAM ĐOAN**

Tôi cam kết báo cáo này là sản phẩm nghiên cứu và thực hiện của bản thân, không sao chép nội dung từ nguồn khác.

# **TÓM TẮT**

Trong tình hình nhu cầu chăm sóc sức khỏe từ xa ngày càng cao, các hệ thống Telehealth có vai trò quan trọng trong việc liên kết bệnh nhân với bác sĩ, giúp giảm áp lực cho bệnh viện và tăng cường khả năng tiếp cận dịch vụ y tế.

Dự án này phát triển một hệ thống đặt lịch khám bệnh trực tuyến dựa trên cấu trúc MERN Stack (MongoDB, Express, React, Node.js). Hệ thống được xây dựng với ba chức năng chính:

• Bệnh nhân: tạo tài khoản, chọn chuyên khoa, hẹn gặp bác sĩ và theo dõi lịch sử khám. Bác sĩ: đăng nhập, điều hành lịch làm việc, xác nhận cuộc hẹn và lập hồ sơ bệnh án điện tử.

• Quản trị viên (Admin): điều hành người dùng, chuyên ngành, cơ sở y tế và theo dõi báo cáo thống kê.

Phần backend được xây dựng trên Node.js + Express, kết nối với cơ sở dữ liệu MongoDB thông qua các schema rõ ràng: User, Doctor, Patient, Appointment, Schedule, MedicalRecord. Phần giao diện người dùng được phát triển bằng React, cung cấp sự thân thiện và khả năng tương tác thời gian thực với API.

Hệ thống đã trải qua kiểm tra toàn bộ với các tình huống thực tế: lên lịch hẹn, quản lý hồ sơ bệnh nhân, phân tích số liệu. Kết quả chỉ ra rằng ứng dụng vận hành ổn định, thực hiện chính xác các nghiệp vụ và có tiềm năng mở rộng trong tương lai.

Dự án không chỉ tăng cường hiểu biết về lập trình web full-stack, mà còn chứng minh khả năng áp dụng công nghệ trong lĩnh vực y tế, góp phần hiện đại hóa quy trình khám chữa bệnh và nâng cao chất lượng chăm sóc sức khỏe cộng đồng

Từ khóa: Telehealth, MERN Stack, Đặt lịch khám bệnh, Hệ thống y tế trực tuyến.

# **DANH MỤC BẢNG VÀ HÌNH VẼ**

[Hình 1 Kiến trúc hệ thống Telehealth 13](#_Toc207479905)

[Hình 2 Sơ đồ Use Case 14](#_Toc207479906)

[Hình 3 Sơ đồ quan hệ ERD 17](#_Toc207479907)

[Hình 4 Sequence Diagram Patient Flow đặt lịch khám 19](#_Toc207479908)

[Hình 5 Giao diện UI cơ bản 70](#_Toc207479909)

[Bảng 1 Quyền hạn của user 10](#_Toc207480325)

[Bảng 2 Routes chính 36](#_Toc207480326)

[Bảng 3 Kết quả seed dữ liệu test 59](#_Toc207480327)

[Bảng 4 TestCase Patient - Doctor - Admin Flow 61](#_Toc207480328)

**DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT**

* **API** (Application Programming Interface): Giao diện lập trình ứng dụng, cho phép frontend và backend trao đổi dữ liệu.
* **BE** (Backend): Phần xử lý nghiệp vụ phía máy chủ, xây dựng bằng Node.js + Express.
* **CRUD** (Create, Read, Update, Delete): Các thao tác cơ bản với dữ liệu.
* **DB** (Database): Cơ sở dữ liệu.
* **E2E** (End-to-End): Kiểm thử toàn diện từ đầu đến cuối hệ thống.
* **FE** (Frontend): Phần giao diện người dùng, xây dựng bằng React.
* **HTTP** (HyperText Transfer Protocol): Giao thức truyền tải siêu văn bản.
* **HTTPS** (HTTP Secure): Phiên bản bảo mật của HTTP.
* **ID** (Identifier): Định danh duy nhất cho một thực thể (User, Appointment, Doctor…).
* **JSON** (JavaScript Object Notation): Định dạng dữ liệu trao đổi giữa frontend và backend.
* **JWT** (JSON Web Token): Chuẩn token dùng để xác thực và phân quyền.
* **MERN Stack**: Bộ công nghệ gồm MongoDB, Express, React, Node.js.
* **MVC** (Model – View – Controller): Mô hình kiến trúc phần mềm phân tách dữ liệu, giao diện và xử lý.
* **NoSQL** (Not Only SQL): Kiểu cơ sở dữ liệu phi quan hệ (MongoDB).
* **OTP** (One-Time Password): Mã xác thực dùng một lần.
* **REST API** (Representational State Transfer API): API dựa trên giao thức HTTP, phổ biến trong web service.
* **TTL** (Time To Live): Thời gian sống của dữ liệu, giới hạn vòng đời OTP.
* **UI** (User Interface): Giao diện người dùng.
* **UX** (User Experience): Trải nghiệm người dùng.
* **2FA** (Two-Factor Authentication): Xác thực hai lớp (đa yếu tố).

# **MỤC LỤC**

[**LỜI CAM ĐOAN** 2](#_Toc207481918)

[**TÓM TẮT** 3](#_Toc207481919)

[**DANH MỤC BẢNG VÀ HÌNH VẼ** 4](#_Toc207481920)

[**MỤC LỤC** 6](#_Toc207481921)

[**CHƯƠNG 1: TỔNG QUAN** 8](#_Toc207481922)

[**1.1 Giới thiệu đề tài** 8](#_Toc207481923)

[1.1.1 Nhiệm vụ và mục tiêu 9](#_Toc207481924)

[1.1.2 Phạm vi và giới hạn 9](#_Toc207481925)

[**1.2 Cấu trúc đồ án** 11](#_Toc207481926)

[**CHƯƠNG 2: CƠ SỞ LÝ THUYẾT** 12](#_Toc207481927)

[**Các khái niệm và công nghệ** 12](#_Toc207481928)

[**CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG** 14](#_Toc207481929)

[**3.1 Use Case Diagram** 14](#_Toc207481930)

[**3.2 Mô hình dữ liệu ERD** 15](#_Toc207481931)

[**3.3 Luồng dữ liệu** 18](#_Toc207481932)

[**CHƯƠNG 4: CÀI ĐẶT VÀ TRIỂN KHAI** 20](#_Toc207481933)

[**4.1 Môi trường triển khai** 20](#_Toc207481934)

[**4.2 Cài đặt Backend** 20](#_Toc207481935)

[**4.3 Cài đặt Frontend** 20](#_Toc207481936)

[**4.4 Dữ liệu mẫu (Seed Script)** 21](#_Toc207481937)

[**CHƯƠNG 5: KIỂM THỬ HỆ THỐNG VÀ ĐÁNH GIÁ** 22](#_Toc207481938)

[**5.1 Phương pháp kiểm thử** 22](#_Toc207481939)

[**5.2 Kịch bản kiểm thử** 22](#_Toc207481940)

[**5.3 Kết quả kiểm thử** 22](#_Toc207481941)

[**CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN** 23](#_Toc207481942)

[**6.1 Kết luận** 23](#_Toc207481943)

[**6.2 Hạn chế** 24](#_Toc207481944)

[**6.3 Hướng phát triển** 24](#_Toc207481945)

[**TÀI LIỆU THAM KHẢO** 26](#_Toc207481946)

[**PHỤ LỤC** 27](#_Toc207481947)

# **CHƯƠNG 1: TỔNG QUAN**

## **1.1 Giới thiệu đề tài**

* **Bối cảnh:**
  + Hiện nay, tình hình y tế cho thấy nhu cầu đặt lịch khám trực tuyến đã tăng. [3] Grand View Research cho hay thị trường y tế từ xa toàn cầu có giá trị 83.5 tỷ đô la nhưng dự kiến tăng lên 455.27 tỷ đô la vào năm 2030 với bình quân tốc độ tăng trưởng hàng năm là 24.68%.
  + Telehealth giúp giảm tải phòng khám, tiết kiệm thời gian cho cả hai bên. Ở Việt Nam, Docosan [1][6] ICTVietnam, 2020; TechCrunch, 2021, Vinmec Telehealth và Viettel [4][5] Telehealth Tuổi Trẻ, 2020; Viettel Solutions, 2020 đã triển khai thành công khám bệnh từ xa, đạt hiệu quả cao tại nông thôn, miền núi và hải đảo.
  + Trên thị trường những báo cáo về nghiên cứu đã chỉ ra rằng đây là xu hướng hoạt động y tế toàn cầu. [2] J.P. Morgan cho rằng trong môi trường thanh khoản, công nghệ hiện đại, sự tăng cường nhu cầu sức khỏe và cải cách chính sách, Y tế từ xa trở thành một trong ba trụ cột tham gia liên kết chăm sóc sức khỏe.
* **Lý do hình thành đề tài:**
  + Ở nước ta, việc lập lịch bằng hình thức online chưa phổ biến tại nhiều cơ sở y tế, đặc biệt là các phòng khám tư nhân cũng như bệnh viện tuyến dưới.
  + Cần một hệ thống dễ sử dụng, bảo mật, quản lý lịch rảnh bác sĩ, lịch hẹn bệnh nhân, giúp nâng cao trải nghiệm và hiệu quả khám chữa bệnh.
* **Ý nghĩa khoa học và thực tiễn:**
  + Khoa học: Áp dụng MERN stack, JWT, REST API, thiết kế cơ sở dữ liệu, kiểm soát luồng thông tin giữa bệnh nhân, bác sĩ và admin, nghiên cứu quản lý lịch hẹn và phân quyền
  + Thực tiễn: Giúp bệnh nhân đặt lịch nhanh chóng, bác sĩ quản lý lịch cá nhân hiệu quả, admin theo dõi, thống kê và quản lý dữ liệu khám chữa bệnh toàn hệ thống

### 1.1.1 Nhiệm vụ và mục tiêu

* Mục tiêu:
  + Xây dựng hệ thống Telehealth cho bệnh nhân, bác sĩ và admin.
  + Hỗ trợ chức năng đặt lịch, quản lý lịch hẹn, phân quyền user.
* Nhiệm vụ:
  + Thiết kế và triển khai backend (Node.js + Express + MongoDB).
  + Thiết kế frontend (React) hiển thị lịch, form đặt lịch, dashboard.
  + Tạo database, seed dữ liệu test.
  + Bảo mật: xác thực JWT, phân quyền.
  + Xử lý lịch rảnh bác sĩ theo khung giờ hành chính

### 1.1.2 Phạm vi và giới hạn

* Đối tượng: bệnh nhân, bác sĩ, admin.
* Công nghệ sử dụng: MERN stack (MongoDB, Express, React, Node.js), JWT, REST API.
* Giới hạn:
  + Chỉ quản lý khung giờ khám hành chính (8h - 11h, 13h - 17h).
  + Không triển khai thanh toán online, tư vấn video call.
  + Không bao gồm tính năng chat trực tiếp hoặc thông báo push.

Bảng 1 Quyền hạn của user

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | **Chức năng / Module** | **Quyền hạn cụ thể** | **Ghi chú** |  |
| Patient | Hồ sơ cá nhân / Profile | Xem, cập nhật thông tin cơ bản (User, Patient) | Không được thay đổi role, email đăng ký tài khoản |  |
|  | Lịch khám / Appointment | Đặt lịch, xem lịch đã đặt, hủy lịch nếu chưa confirm | OTP xác thực |  |
|  | Hồ sơ y tế / Medical Record | Xem hồ sơ y tế của chính mình | Không được sửa, chỉ đọc |  |
|  | OTP | Nhận và xác thực mã OTP để đặt lịch | TTL giới hạn spam |  |
| Doctor | Hồ sơ cá nhân / Profile | Xem, cập nhật thông tin cơ bản (User, Doctor) | Không được thay đổi role, email đăng ký tài khoản, chuyên khoa, cơ sở y tế |  |
|  | Lịch làm việc / Schedule | Quản lý lịch làm việc cá nhân, tạo request | Theo khung giờ hành chính cố định |  |
|  | Lịch khám / Appointment | Xem danh sách lịch khám của bản thân | Không được tạo lịch cho bệnh nhân khác |  |
|  | Hồ sơ y tế / Medical Record | Tạo hồ sơ, cập nhật, kết luận hồ sơ y tế cho bệnh nhân | Snapshot doctor, patient |  |
| Admin | Quản lý User | Patient / Doctor / User CRUD | Có thể phân quyền |  |
|  | Quản lý Location & Specialty | Tạo, cập nhật, xóa Locations và Specialties | Cascade xóa chuyên khoa khi xóa location |  |
|  | Quản lý Appointment | Xem tất cả lịch khám, trạng thái của các doctor/patient | Chỉnh sửa nếu cần |  |
|  | Quản lý Medical Record | Xem toàn bộ Medical Records | Không sửa snapshot lịch sử nếu muốn giữ integrity |  |
|  | OTP | Quản lý OTP test, debug | Optional |  |

## **1.2 Cấu trúc đồ án**

* Chương 1: Tổng quan về đề tài, nhiệm vụ, phạm vi, cấu trúc đồ án.
* Chương 2: Cơ sở lý thuyết và mô hình, bao gồm công nghệ, mô hình dữ liệu, luồng dữ liệu, thiết kế sơ đồ quan hệ ERD, wireframe.
* Chương 3: Kết quả thực nghiệm: thiết kế backend/frontend, test dữ liệu, giao diện, kết quả test.
* Chương 4: Kết luận và kiến nghị, mở rộng, cải tiến.

# **CHƯƠNG 2: CƠ SỞ LÝ THUYẾT**

## **Các khái niệm và công nghệ**

Kiến trúc hệ thống

Hệ thống Telehealth được xây dựng theo kiến trúc MERN stack, bao gồm:

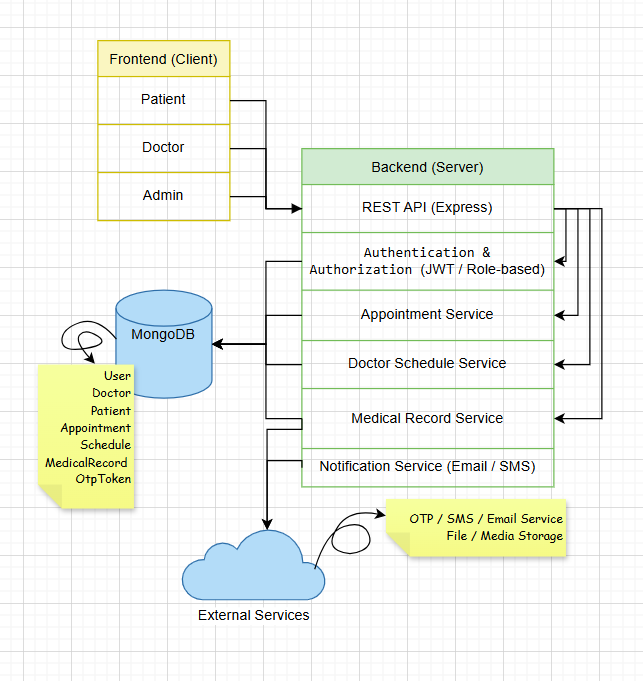
* MongoDB: Cơ sở dữ liệu NoSQL, lưu trữ thông tin người dùng, bác sĩ, bệnh nhân, lịch hẹn, chuyên khoa, địa điểm, hồ sơ y tế
* Express.js: Framework cho backend Node.js, xử lý REST API, routing, middleware.
* React.js: Frontend, xây dựng giao diện người dùng, hiển thị dữ liệu từ API.
* Node.js: Runtime cho server, xử lý logic backend.

Xác thực và bảo mật:

* Sử dụng JWT (JSON Web Token) để xác thực người dùng, phân quyền theo role (patient, doctor, admin).
* Middleware verifyToken kiểm tra quyền truy cập trước khi thực hiện các API quan trọng.

Giao tiếp giữa frontend và backend:

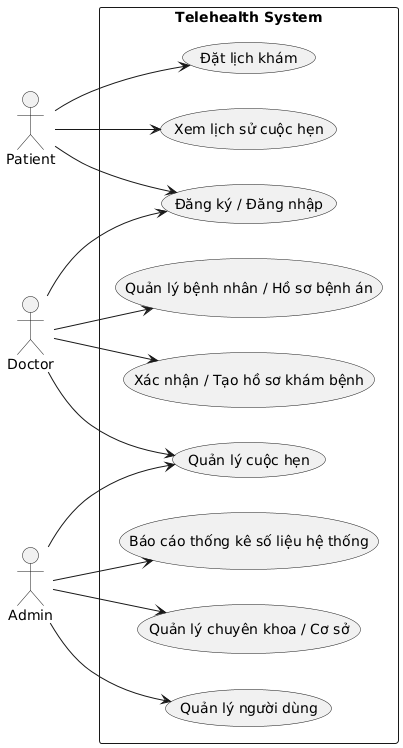
* REST API: frontend gọi API thông qua HTTP (GET, POST, PUT, DELETE).
* Bearer token: token JWT được gửi trong header để xác thực.



Hình 1 Kiến trúc hệ thống Telehealth

# **CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

## **3.1 Use Case Diagram**



Hình 2 Sơ đồ Use Case

**Mô tả chi tiết các Use Case**

* Patient
  + Đăng ký / Đăng nhập: cho phép bệnh nhân tạo tài khoản hoặc đăng nhập hệ thống.
  + Đặt lịch khám: bệnh nhân lựa chọn bác sĩ, thời gian rảnh và tạo cuộc hẹn trực tuyến
  + Xem lịch sử cuộc hẹn: theo dõi lại các cuộc hẹn đã đặt trước đó.
* Doctor
  + Đăng ký / Đăng nhập: cho phép bác sĩ đăng nhập và sử dụng hệ thống.
  + Quản lý cuộc hẹn: xem, sắp xếp và xác nhận lịch hẹn với bệnh nhân.
  + Xác nhận / Tạo hồ sơ khám bệnh: xác nhận thông tin khám và lập hồ sơ bệnh án.
  + Quản lý bệnh nhân / Hồ sơ bệnh án: quản lý thông tin bệnh nhân và cập nhật hồ sơ.
* Admin
  + Quản lý người dùng: tạo, khóa, phân quyền tài khoản bệnh nhân và bác sĩ.
  + Quản lý chuyên khoa / Cơ sở: thêm, sửa, xóa thông tin chuyên khoa và cơ sở y tế.
  + Báo cáo thống kê số liệu hệ thống: tổng hợp dữ liệu và xuất báo cáo.

## **3.2 Mô hình dữ liệu ERD**

Hệ thống có các bảng (collection) chính:

* User
  + Thông tin cơ bản: name, email, password, role (patient, doctor, admin).
  + Quan hệ: doctor ↔ user, patient ↔ user.
* Doctor
  + Liên kết với User.
  + Liên kết với Specialty (chuyên khoa) và Location (phòng khám).
  + Lưu thông tin lịch rảnh, lịch hẹn.
* Patient
  + Liên kết với User.
  + Lưu lịch hẹn, lịch sử khám bệnh.
* Appointment
  + Lưu thông tin ngày giờ, trạng thái (pending, confirmed, completed, canceled, expired), bác sĩ, bệnh nhân.
  + Kiểm soát lịch rảnh bác sĩ dựa trên khung giờ cố định và lịch hẹn hiện tại.
* Specialty
  + Chuyên khoa, liên kết với Location.
  + Quy tắc: không tạo specialty “chung” trước, phải gán location khi tạo
* Location
  + Địa điểm phòng khám.
  + Liên kết với các chuyên khoa và bác sĩ.

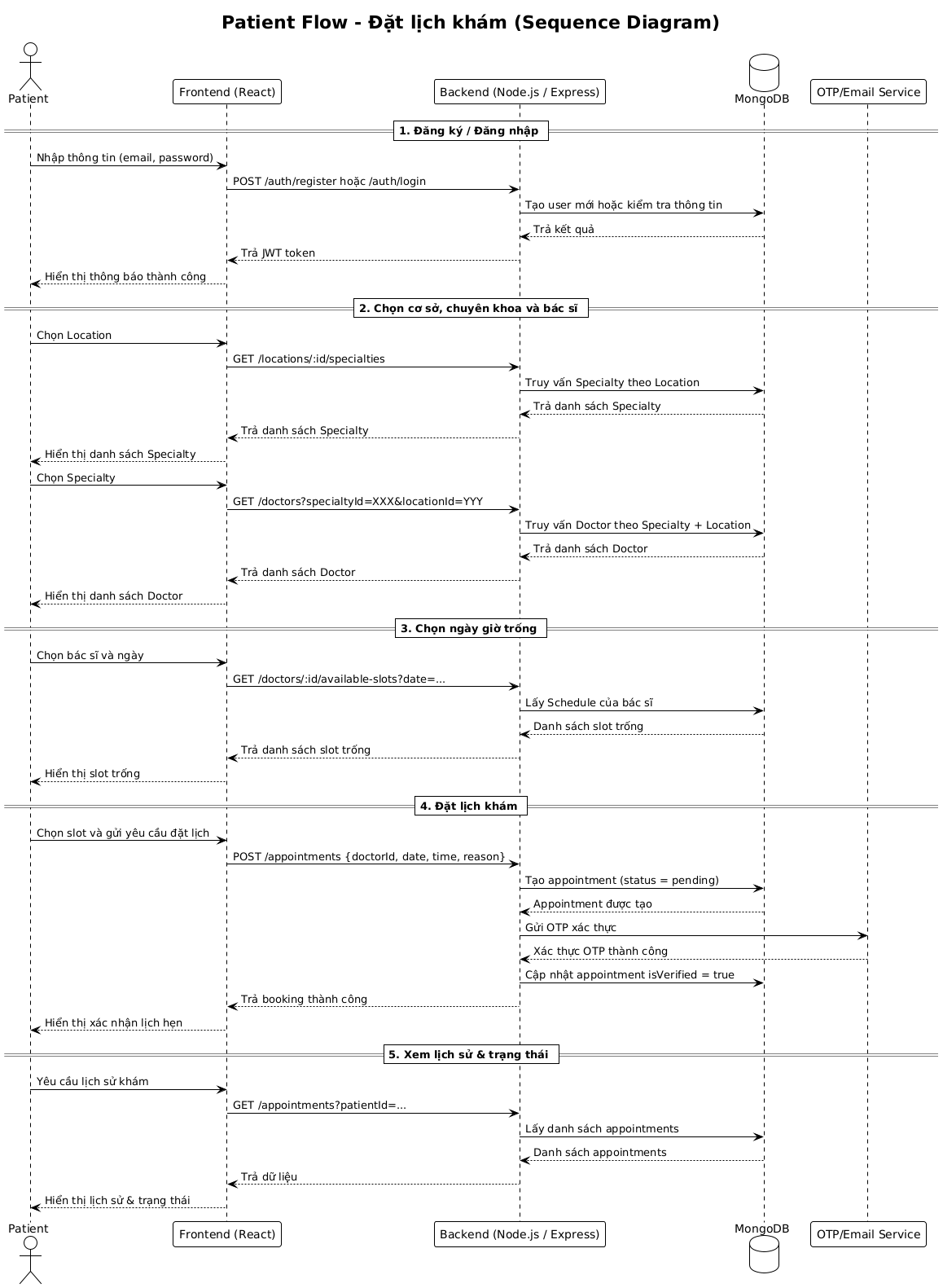
****

Hình 3 Sơ đồ quan hệ ERD

## **3.3 Luồng dữ liệu**

Luồng chính của hệ thống:

* Patient Flow
  + Đăng ký / đăng nhập.
  + Chọn bác sĩ hoặc chuyên khoa.
  + Chọn ngày giờ trống, đặt lịch.
  + Xem lịch sử khám và trạng thái lịch hẹn.
* Doctor Flow
  + Đăng nhập.
  + Xem lịch cá nhân theo ngày/tuần.
  + Quản lý lịch hẹn: xác nhận, hủy, hoàn tất.
* Admin Flow
  + Quản lý người dùng: patient, doctor, admin.
  + Quản lý chuyên khoa và phòng khám.
  + Xem báo cáo thống kê cơ bản (số lượng bệnh nhân, bác sĩ, lịch hẹn).



Hình 4 Sequence Diagram Patient Flow đặt lịch khám

# **CHƯƠNG 4: CÀI ĐẶT VÀ TRIỂN KHAI**

## **4.1 Môi trường triển khai**

Hệ thống được phát triển theo kiến trúc **MERN Stack** gồm:

* **Backend**: Node.js, Express.js, MongoDB.
* **Frontend**: React.js với thư viện giao diện hỗ trợ (TailwindCSS).
* **Công cụ phát triển**: Visual Studio Code, Postman, MongoDB.

## **4.2 Cài đặt Backend**

Phần backend bao gồm các thành phần chính:

* **Models**: định nghĩa cấu trúc dữ liệu cho User, Doctor, Patient, Appointment, Schedule, MedicalRecord.
* **Controllers**: xử lý logic nghiệp vụ (đặt lịch, xem lịch, quản lý hồ sơ bệnh án, thống kê).
* **Routes**: ánh xạ các API endpoint tới controller tương ứng.

Ghi chú: Chi tiết code được trình bày trong [Phụ lục A.](#_PHỤ_LỤC)

## **4.3 Cài đặt Frontend**

Frontend được xây dựng trên **React**, tổ chức thành các component tương ứng với từng nhóm người dùng (Patient, Doctor, Admin).  
Các chức năng chính:

* Đăng nhập/Đăng ký.
* Đặt lịch khám (Patient).
* Xem lịch làm việc (Doctor).
* Quản lý thống kê và danh mục (Admin).

Chi tiết cài đặt được trình bày trong [Phụ lục B**.**](#_PHỤ_LỤC)

## **4.4 Dữ liệu mẫu (Seed Script)**

Để phục vụ quá trình thử nghiệm, hệ thống có script **seed.js** sinh dữ liệu ban đầu gồm:

* 01 tài khoản quản trị hệ thống.
* 05 bác sĩ thuộc các chuyên khoa khác nhau.
* 10 bệnh nhân đăng ký hệ thống.
* 20 lịch hẹn phân bố trong tuần hiện tại.

Nội dung script và dữ liệu mẫu được trình bày trong[Phụ lục C**.**](#_PHỤ_LỤC)

# **CHƯƠNG 5: KIỂM THỬ HỆ THỐNG VÀ ĐÁNH GIÁ**

## **5.1 Phương pháp kiểm thử**

Hệ thống được kiểm thử tập trung vào việc xác minh chức năng và hành vi đầu ra dựa trên đầu vào.  
Các nhóm người dùng được kiểm thử: Bệnh nhân, Bác sĩ, Quản trị viên.

## **5.2 Kịch bản kiểm thử**

* Patient Flow: đăng nhập, đặt lịch, xem lịch khám, xem hồ sơ bệnh án.
* Doctor Flow: đăng nhập, xem lịch làm việc theo ngày/tuần, tạo / quản lý hồ sơ bệnh án.
* Admin Flow: quản lý bác sĩ, chuyên khoa, cơ sở, thống kê hệ thống.

Bảng test case chi tiết được trình bày trong [Phụ lục D.](#_PHỤ_LỤC)

## **5.3 Kết quả kiểm thử**

Kết quả cho thấy các chức năng cốt lõi đều hoạt động ổn định:

* Đặt lịch và hủy lịch thành công.
* Hệ thống tự động loại bỏ các khung giờ đã được đặt.
* Bác sĩ có thể xem lịch làm việc và tạo / quản lý hồ sơ bệnh án.
* Quản trị viên quản lý người dùng, cơ sở / chuyên khoa, thống kê hệ thống.

# **CHƯƠNG 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

## **6.1 Kết luận**

Sau quá trình nghiên cứu và triển khai, nhóm đã xây dựng thành công hệ thống Telehealth System – Hệ thống đặt lịch khám bệnh trực tuyến trên nền tảng MERN stack. Hệ thống cơ bản đáp ứng được các yêu cầu đề ra, với các điểm nổi bật:

* Triển khai thành công nền tảng MERN stack:
  + Backend sử dụng Node.js + Express + MongoDB, thiết kế mô hình dữ liệu hợp lý, triển khai đầy đủ các controller, route và middleware xác thực – phân quyền.
  + Frontend xây dựng bằng React.js, tận dụng component tái sử dụng, giao tiếp API thông qua cơ chế Bearer token, đảm bảo trải nghiệm người dùng liên tục.
* Chức năng đặt lịch và quản lý người dùng:
  + Bệnh nhân có thể đăng ký tài khoản, lựa chọn bác sĩ/chuyên khoa, đặt lịch khám, theo dõi lịch sử khám, xem lại hồ sơ bệnh án của mình.
  + Bác sĩ có khả năng quản lý lịch hẹn cá nhân, quản lý hồ sơ bệnh án, xác nhận hoặc hủy lịch khi cần.
  + Admin quản lý toàn bộ người dùng, chuyên khoa, phòng khám và có thể theo dõi thống kê hệ thống.
* Quản lý lịch rảnh và phân quyền:
  + Hệ thống tự động sinh khung giờ hành chính (08:00–11:00, 13:00–17:00) và chỉ hiển thị các slot còn trống cho bệnh nhân.
  + Phân quyền rõ ràng giữa patient – doctor – admin, đảm bảo an toàn dữ liệu và tính riêng tư.
* Kiểm thử và dữ liệu demo:
  + Seed dữ liệu thử nghiệm: 10–15 bác sĩ, 20–30 bệnh nhân, 50 lịch hẹn.
  + Thực hiện kiểm thử end-to-end cho thấy hệ thống hoạt động ổn định, quy trình đặt lịch – xác nhận – quản lý dữ liệu được vận hành trơn tru, không phát sinh lỗi nghiêm trọng.

Tổng thể, hệ thống chứng minh tính khả thi của giải pháp công nghệ web trong việc hỗ trợ quản lý hoạt động khám chữa bệnh trực tuyến, góp phần nâng cao hiệu quả điều phối và tối ưu trải nghiệm của cả bệnh nhân lẫn bác sĩ.

## **6.2 Hạn chế**

* Bên cạnh những kết quả đạt được, hệ thống vẫn còn một số hạn chế:
  + Chưa tích hợp chức năng thanh toán trực tuyến, việc đặt lịch chỉ dừng ở mức đăng ký và xác nhận.
  + Giao diện người dùng (UI/UX) còn ở mức cơ bản, chưa tối ưu cho trải nghiệm trên thiết bị di động.
  + Chưa triển khai tư vấn từ xa (telemedicine/video call)
  + Chưa được thử nghiệm trên dữ liệu thực tế từ phòng khám/bệnh viện nên tính ứng dụng thực tiễn cần được kiểm chứng thêm.

## **6.3 Hướng phát triển**

Trong thời gian tới, hệ thống có thể mở rộng và cải tiến theo các hướng sau:

* Mở rộng tính năng:
  + Tích hợp thanh toán trực tuyến qua ví điện tử, ngân hàng.
  + Bổ sung video call/teleconsultation để bác sĩ tư vấn trực tiếp từ xa.
  + Xây dựng thống kê nâng cao: doanh thu, lượt khám, hiệu suất bác sĩ.
* Cải thiện trải nghiệm người dùng:
  + Thiết kế responsive UI/UX cho thiết bị di động và tablet.
  + Tối ưu hóa form đặt lịch và giao diện hiển thị lịch khám trực quan hơn.
* Nâng cao hiệu năng backend:
  + Tối ưu hóa query MongoDB, áp dụng cache cho dữ liệu truy xuất thường xuyên.
  + Sử dụng service worker và cơ chế tối ưu front-end nhằm giảm tải server.
* Bảo mật dữ liệu:
  + Mã hóa thông tin nhạy cảm.
  + Áp dụng đăng nhập đa yếu tố (2FA) cho bác sĩ và admin.
  + Kiểm thử bảo mật để phát hiện lỗ hổng.
* Mở rộng hệ thống:
  + Thêm nhiều loại tài khoản chuyên biệt như y tá, nhân viên lễ tân.
  + Hỗ trợ quản lý nhiều phòng khám/chi nhánh trên cùng một nền tảng.
  + Trong tương lai xa, tích hợp Chatbot AI hỗ trợ tư vấn cơ bản, chẩn đoán sơ bộ dựa trên dữ liệu bệnh án.

# **TÀI LIỆU THAM KHẢO**

[1] ICTVietnam. (2020). Startup Telehealth – kênh chăm sóc sức khỏe hữu hiệu mùa dịch COVID-19. ICTVietnam. Truy cập từ: [https://ictvietnam.vn/startup-telehealth-kenh-cham-soc-suc-khoe-huu-hieu-mua-dich-covid-19-25917.html](https://ictvietnam.vn/startup-telehealth-kenh-cham-soc-suc-khoe-huu-hieu-mua-dich-covid-19-25917.html?utm_source=chatgpt.com)

[2] J.P. Morgan. (2023). Telehealth and digitalized healthcare. J.P. Morgan Payments Unbound. Truy cập từ: [https://www.jpmorgan.com/payments/paymentsunbound/volume-2/telehealth-and-digitalized-healthcare](https://www.jpmorgan.com/payments/paymentsunbound/volume-2/telehealth-and-digitalized-healthcare?utm_source=chatgpt.com)

[3] Grand View Research. (2022). Telehealth Market Size, Share & Trends Analysis Report By Component, By Mode of Delivery, By Application, By End-use, By Region, And Segment Forecasts, 2022 – 2030. Grand View Research. Truy cập từ: <https://www.grandviewresearch.com/press-release/global-telehealth-market>

[4] Tuổi Trẻ. (2020). Bộ Y tế khai trương 1000 cơ sở khám chữa bệnh từ xa với giải pháp Viettel Telehealth. Báo Tuổi Trẻ. Truy cập từ: [https://tuoitre.vn/bo-y-te-khai-truong-1000-co-so-kham-chua-benh-tu-xa-voi-giai-phap-viettel-telehealth-20201001144925254.htm](https://tuoitre.vn/bo-y-te-khai-truong-1000-co-so-kham-chua-benh-tu-xa-voi-giai-phap-viettel-telehealth-20201001144925254.htm?utm_source=chatgpt.com)

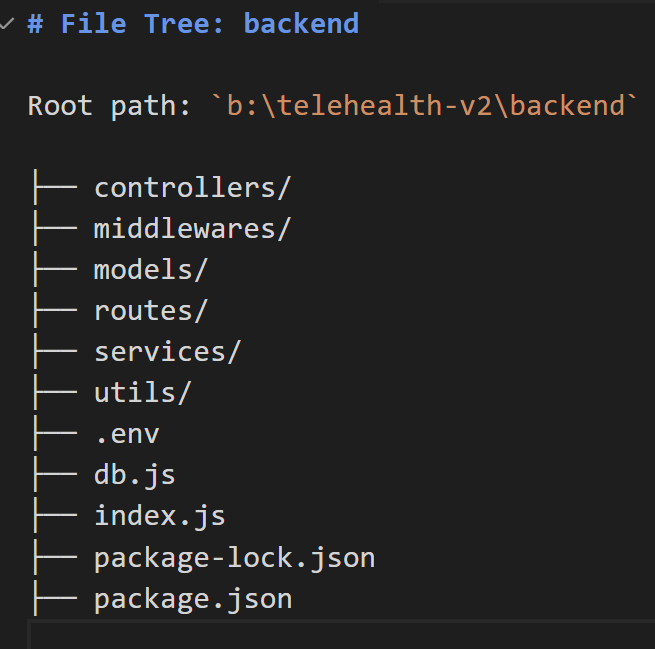
[5] Viettel Solutions. (2020). Giải pháp Telehealth của Viettel Solutions được quốc tế vinh danh. Viettel Solutions. Truy cập từ: [https://mic.gov.vn/giai-phap-telehealth-cua-viettel-solutions-duoc-quoc-te-vinh-danh-197147260.htm](https://mic.gov.vn/giai-phap-telehealth-cua-viettel-solutions-duoc-quoc-te-vinh-danh-197147260.htm?utm_source=chatgpt.com)

[6] TechCrunch. (2021). Vietnam-based healthcare booking app Docosan gets $1M seed funding led by AppWorks. TechCrunch. Truy cập từ: [https://techcrunch.com/2021/04/13/vietnam-based-healthcare-booking-app-docosan-gets-1m-seed-funding-led-by-appworks/](https://techcrunch.com/2021/04/13/vietnam-based-healthcare-booking-app-docosan-gets-1m-seed-funding-led-by-appworks/?utm_source=chatgpt.com)

# **PHỤ LỤC**

**Phụ lục A – Code Backend (controllers, routes, models)**

**A.1. Cấu trúc thư mục chínnh**



**A.2 Models**

* **User.js**
* const mongoose = require('mongoose');
* const bcrypt = require('bcrypt');
* const Patient = require('./Patient');
* const Doctor = require('./Doctor');
* const Specialty = require('./Specialty');
* const SALT\_ROUNDS = 10;
* const userSchema = new mongoose.Schema(
* {
* fullName: {
* type: String,
* required: [true, 'Vui lòng nhập họ tên'],
* trim: true,
* minlength: [2, 'Họ tên quá ngắn'],
* maxlength: [100, 'Họ tên quá dài'],
* },
* email: {
* type: String,
* unique: true,
* required: [true, 'Vui lòng nhập email'],
* lowercase: true,
* trim: true,
* match: [/.+@.+\..+/, 'Email không hợp lệ'],
* },
* password: {
* type: String,
* required: [true, 'Vui lòng nhập mật khẩu'],
* minlength: [6, 'Mật khẩu phải từ 6 ký tự trở lên'],
* },
* role: {
* type: String,
* enum: ['patient', 'doctor', 'admin'],
* required: true,
* default: 'patient',
* },
* birthYear: {
* type: Number,
* min: 1900,
* max: new Date().getFullYear(),
* default: null,
* },
* gender: {
* type: String,
* enum: ['male', 'female', 'other'],
* default: 'other',
* },
* phone: {
* type: String,
* trim: true,
* match: [/^$|^[0-9]{10}$/, 'Số điện thoại không hợp lệ'],
* default: '',
* },
* avatar: {
* type: String,
* trim: true,
* default: '',
* },
* bio: {
* type: String,
* trim: true,
* default: '',
* maxlength: [2000, 'Bio quá dài'],
* },
* specialty: {
* type: mongoose.Schema.Types.ObjectId,
* ref: 'Specialty',
* },
* location: {
* type: mongoose.Schema.Types.ObjectId,
* ref: 'Location',
* },
* },
* { timestamps: true }
* );
* // Hash mật khẩu trước khi lưu
* userSchema.pre('save', async function (next) {
* try {
* if (this.isModified('email')) this.email = String(this.email).trim().toLowerCase();
* if (this.isNew || this.isModified('password')) {
* const cleanPassword = String(this.password).trim();
* const isAlreadyHashed = cleanPassword.startsWith('$2b$');
* if (!isAlreadyHashed) {
* this.password = await bcrypt.hash(cleanPassword, SALT\_ROUNDS);
* } else this.password = cleanPassword;
* }
* next();
* } catch (err) {
* next(err);
* }
* });
* // So sánh mật khẩu
* userSchema.methods.comparePassword = async function (candidatePassword) {
* return bcrypt.compare(String(candidatePassword).trim(), this.password);
* };
* // Loại bỏ mật khẩu khi trả về JSON
* userSchema.methods.toJSON = function () {
* const obj = this.toObject();
* delete obj.password;
* return obj;
* };
* // Post-save hook: Sau khi lưu tự động tạo Patient/Doctor nếu chưa có
* userSchema.post('save', async function (doc, next) {
* try {
* if (doc.role === 'patient') {
* const existingPatient = await Patient.findOne({ user: doc.\_id });
* if (!existingPatient) await Patient.create({ user: doc.\_id });
* } else if (doc.role === 'doctor') {
* const existingDoctor = await Doctor.findOne({ user: doc.\_id });
* if (!existingDoctor) {
* const doctorDoc = await Doctor.create({
* user: doc.\_id,
* specialty: doc.specialty,
* location: doc.location,
* });
* // Chỉ cập nhật Specialty, không update Location nữa
* if (doc.specialty) {
* const spec = await Specialty.findById(doc.specialty);
* if (spec) {
* spec.doctors = spec.doctors || [];
* if (!spec.doctors.includes(doc.\_id)) spec.doctors.push(doc.\_id);
* await spec.save();
* }
* }
* }
* }
* next();
* } catch (err) {
* // TODO: Xử lý logging chi tiết hơn khi triển khai production
* next();
* }
* });
* module.exports = mongoose.model('User', userSchema);
* **Appointment.js**
* const mongoose = require('mongoose');
* const dayjs = require('dayjs');
* const utc = require('dayjs/plugin/utc');
* const timezone = require('dayjs/plugin/timezone');
* const { toVN } = require('../utils/timezone');
* dayjs.extend(utc);
* dayjs.extend(timezone);
* const APPOINTMENT\_STATUSES = ['pending', 'confirmed', 'cancelled', 'completed', 'expired'];
* const appointmentSchema = new mongoose.Schema(
* {
* location: { type: mongoose.Schema.Types.ObjectId, ref: 'Location', required: true },
* specialty: { type: mongoose.Schema.Types.ObjectId, ref: 'Specialty', required: true },
* doctor: { type: mongoose.Schema.Types.ObjectId, ref: 'Doctor', required: true },
* patient: { type: mongoose.Schema.Types.ObjectId, ref: 'Patient', required: true },
* datetime: {
* type: Date,
* required: true,
* validate: {
* validator: function (value) {
* if (!this.isNew && !this.isModified('datetime')) return true;  // nếu không phải tạo mới và không sửa datetime thì bỏ validate
* return value && value > new Date();
* },
* message: 'Không thể đặt lịch ở thời điểm quá khứ'
* }
* },
* // Auto hiển thị giờ VN
* date: { type: String },
* time: { type: String },
* reason: { type: String, default: '' },
* otp: { type: String },
* otpExpiresAt: { type: Date },
* isVerified: { type: Boolean, default: false },
* status: { type: String, enum: APPOINTMENT\_STATUSES, default: 'pending' },
* workScheduleId: { type: mongoose.Schema.Types.ObjectId, ref: 'Schedule' },
* },
* { timestamps: true }
* );
* // Middleware: tự động set date/time string VN khi lưu
* appointmentSchema.pre('save', function (next) {
* if (this.isModified('datetime') && this.datetime) {
* const vnTime = toVN(this.datetime);
* this.date = dayjs(vnTime).format('YYYY-MM-DD');
* this.time = dayjs(vnTime).format('HH:mm');
* }
* next();
* });
* // Middleware: validate + tự động set date/time khi update
* appointmentSchema.pre('findOneAndUpdate', function (next) {
* const update = this.getUpdate();
* if (update.datetime) {
* // Kiểm tra không cho phép update về quá khứ
* if (update.datetime < new Date()) {
* return next(new Error('Không thể đặt lịch ở thời điểm quá khứ'));
* }
* // Auto set date/time VN
* const vnTime = toVN(update.datetime);
* update.date = dayjs(vnTime).format('YYYY-MM-DD');
* update.time = dayjs(vnTime).format('HH:mm');
* }
* next();
* });
* module.exports = mongoose.model('Appointment', appointmentSchema);

**A.3 Controllers**

* **appointment.controller.js**
* // ===== Xử lý Dropdown APIs =====
* // Lấy danh sách cơ sở y tế
* const getLocations = async (req, res) => {
* try {
* const locations = await Location.find({}, 'name');
* return responseSuccess(res, 'Danh sách cơ sở', { count: locations.length, locations });
* } catch (err) {
* console.error('Lỗi [getLocations]:', err);
* return responseError(res, 'Lỗi khi lấy danh sách cơ sở', 500, err);
* }
* };
* // Lấy danh sách chuyên khoa thuộc cơ sở y tế
* const getSpecialtiesByLocation = async (req, res) => {
* try {
* const { locationId } = req.query;
* if (!locationId || !isValidObjectId(locationId)) {
* return responseError(res, 'ID cơ sở không hợp lệ', 400);
* }
* const specialties = await Specialty.find({ location: locationId })
* .sort({ name: 1 })
* .populate('location', 'name');
* return responseSuccess(res, 'Danh sách chuyên khoa', { count: specialties.length, specialties });
* } catch (err) {
* console.error('Lỗi [getSpecialtiesByLocation]:', err);
* return responseError(res, 'Lỗi khi lấy danh sách chuyên khoa', 500, err);
* }
* };
* // Lấy danh sách bác sĩ thuộc chuyên khoa + cơ sở y tế
* const getDoctorsBySpecialtyAndLocation = async (req, res) => {
* try {
* const { specialtyId, locationId } = req.query;
* if (!specialtyId || !locationId || !isValidObjectId(specialtyId) || !isValidObjectId(locationId)) {
* return responseError(res, 'ID chuyên khoa hoặc cơ sở không hợp lệ', 400);
* }
* const doctors = await Doctor.find({ specialty: specialtyId, location: locationId })
* .populate('user', 'fullName email')
* .populate('specialty', 'name')
* .populate('location', 'name')
* .sort({ \_id: 1 })
* .lean();
* return responseSuccess(res, 'Danh sách bác sĩ', { count: doctors.length, doctors });
* } catch (err) {
* console.error('Lỗi [getDoctorsBySpecialtyAndLocation]:', err);
* return responseError(res, 'Lỗi khi lấy danh sách bác sĩ', 500, err);
* }
* };
* // Lấy slots trống
* const getAvailableSlots = async (req, res) => {
* try {
* const { doctorId, date } = req.query;
* if (!doctorId || !date || !isValidObjectId(doctorId)) {
* return responseError(res, 'Doctor ID hoặc date không hợp lệ', 400);
* }
* const availableSlots = await ScheduleService.getAvailableSlots(doctorId, date);
* return responseSuccess(res, 'Danh sách slot trống', { count: availableSlots.length, availableSlots });
* } catch (err) {
* console.error('Lỗi [getAvailableSlots]:', err);
* return responseError(res, 'Lỗi khi lấy khung giờ trống', 500, err);
* }
* };
* **auth.controller.js**
* // ===== Helper tạo JWT token =====
* const createAccessToken = (user) => {
* if (!process.env.JWT\_SECRET) throw new Error('Thiếu biến môi trường JWT\_SECRET');
* return jwt.sign(
* { id: user.\_id, role: user.role },
* process.env.JWT\_SECRET,
* { expiresIn: process.env.JWT\_EXPIRE || '1d' }
* );
* };
* // --- Helper chuẩn hóa response
* const responseSuccess = (res, message, data = {}) =>
* res.status(200).json({ success: true, message, data });
* const responseError = (res, message, status = 400) =>
* res.status(status).json({ success: false, message });
* // Đăng ký [POST] /api/auth/register
* exports.register = async (req, res) => {
* try {
* const { role, fullName, email, password, specialty, location } = req.body;
* // Validate input cơ bản
* if (!role || !fullName || !email || !password) {
* return responseError(res, 'Vui lòng điền đầy đủ thông tin');
* }
* if (role === 'doctor') {
* if (!specialty || !location) {
* return responseError(res, 'Bác sĩ cần chọn chuyên khoa và cơ sở');
* }
* // Kiểm tra Specialty có thuộc Location không
* const spec = await Specialty.findOne({ \_id: specialty, location: location });
* if (!spec) return responseError(res, 'Chuyên khoa không hợp lệ cho cơ sở y tế này');
* }
* const existingUser = await User.findOne({ email: String(email).trim().toLowerCase() });
* if (existingUser) return responseError(res, 'Email đã tồn tại');
* // Tạo User mới
* const newUser = new User({
* role,
* fullName: fullName.trim(),
* email: String(email).trim().toLowerCase(),
* password: String(password),
* ...(role === 'doctor' && { specialty, location }),
* });
* // Khi save, post-save hook trong User model sẽ tự tạo Patient/Doctor document và cập nhật Specialty/Location
* await newUser.save();
* return responseSuccess(res, 'Đăng ký thành công', {
* id: newUser.\_id,
* fullName: newUser.fullName,
* email: newUser.email,
* role: newUser.role,
* });
* } catch (error) {
* console.error('Lỗi trong register:', error);
* return responseError(res, 'Lỗi server', 500);
* }
* };
* // Đăng nhập [POST] /api/auth/login
* exports.login = async (req, res) => {
* try {
* const { email, password } = req.body;
* if (!email || !password) return responseError(res, 'Vui lòng nhập email và mật khẩu');
* const user = await User.findOne({ email: String(email).trim().toLowerCase() })
* .populate('specialty')
* .populate('location');
* if (!user) return responseError(res, 'Email hoặc mật khẩu không đúng', 401);
* const isMatch = await user.comparePassword(password);
* if (!isMatch) return responseError(res, 'Email hoặc mật khẩu không đúng', 401);
* const token = createAccessToken(user);
* return responseSuccess(res, 'Đăng nhập thành công', {
* token,
* user: {
* id: user.\_id,
* fullName: user.fullName,
* email: user.email,
* role: user.role,
* specialty: user.specialty?.name || null,
* location: user.location?.name || null,
* },
* });
* } catch (error) {
* console.error('Lỗi trong login:', error);
* return responseError(res, 'Lỗi server', 500);
* }
* };

**A.4 Routes**

Bảng 2 Routes chính

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Route** | **Mục đích** | **Quyền truy cập** |
| **User Routes** | | | |
| GET | /api/users | Lấy danh sách tất cả người dùng | Admin |
| GET | /api/users/doctors | Lấy danh sách bác sĩ theo chuyên khoa | Tất cả |
| GET | /api/users/me | Xem thông tin cá nhân | User đã đăng nhập |
| PUT | /api/users/me | Cập nhật thông tin cá nhân | User đã đăng nhập |
| GET | /api/users/:id | Xem thông tin user theo ID | Admin |
| POST | /api/users/ | Tạo user mới | Admin |
| PUT | /api/users/:id | Cập nhật thông tin user theo ID | Admin |
| DELETE | /api/users/:id | Xóa user | Admin |
| **Doctor Routes** | | | |
| GET | /api/doctors/me | Lấy thông tin profile bác sĩ | Doctor |
| PUT | /api/doctors/me | Cập nhật profile bác sĩ | Doctor |
| PUT | /api/doctors/me/password | Thay đổi mật khẩu | Doctor |
| GET | /api/doctors/work-schedule | Lấy lịch làm việc | Doctor |
| GET | /api/doctors/appointments | Lấy tất cả appointment của bác sĩ | Doctor |
| GET | /api/doctors/appointments/date | Lấy appointment theo ngày | Doctor |
| GET | /api/doctors/appointments/:id | Xem chi tiết appointment | Doctor |
| PATCH | /api/doctors/appointments/:id/status | Cập nhật trạng thái appointment | Doctor |
| DELETE | /api/doctors/appointments/:id | Hủy appointment | Doctor |
| POST | /api/doctors/appointments/:id/medical-receipt | Tạo hồ sơ bệnh án cho appointment | Doctor |
| GET | /api/doctors/my-patients | Lấy danh sách bệnh nhân của bác sĩ | Doctor |
| **Patient Routes** | | | |
| GET | /api/patients/me | Xem profile cá nhân | Patient |
| PUT | /api/patients/me | Cập nhật profile cá nhân | Patient |
| PUT | /api/patients/me/password | Thay đổi mật khẩu | Patient |
| GET | /api/patients/me/medical-records | Lấy danh sách hồ sơ bệnh án của chính mình | Patient |
| **Appointment Routes** | | | |
| GET | /api/appointments/locations | Lấy danh sách cơ sở y tế | Tất cả |
| GET | /api/appointments/specialties | Lấy danh sách chuyên khoa theo cơ sở | Tất cả |
| GET | /api/appointments/doctors | Lấy bác sĩ theo chuyên khoa và cơ sở | Tất cả |
| GET | /api/appointments/available-slots | Lấy khung giờ trống của bác sĩ | Tất cả |
| POST | /api/appointments/ | Tạo lịch hẹn mới | Patient |
| POST | /api/appointments/verify-otp | Xác thực OTP khi đặt lịch | Patient |
| POST | /api/appointments/resend-otp | Gửi lại OTP | Patient |
| GET | /api/appointments/ | Lấy danh sách lịch hẹn của user hiện tại | User đã đăng nhập |
| GET | /api/appointments/:id | Xem chi tiết lịch hẹn | User đã đăng nhập |
| DELETE | /api/appointments/:id | Hủy lịch hẹn | User đã đăng nhập |

***// Ghi chú: Chi tiết đầy đủ trong source code đính kèm***

**Phụ lục B – Code Frontend (React)**

**B.1 Cấu trúc thư mục chính**



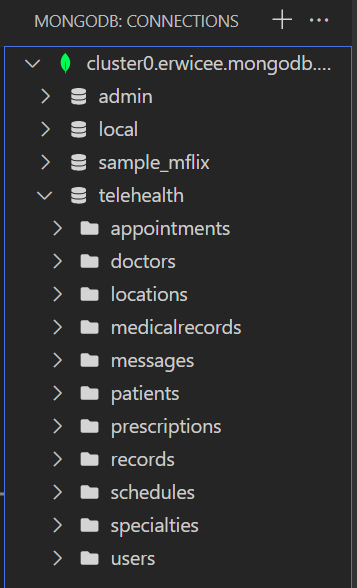
**B.2 Component tiêu biểu**

* **LoginPage.jsx (Trang đăng nhập người dùng)**
* import React, { useState } from 'react';
* import { useNavigate } from 'react-router-dom';
* import './AuthPage.css';
* import authService from '../../services/authService';
* const LoginPage = () => {
* const navigate = useNavigate();
* const [formData, setFormData] = useState({ email: '', password: '' });
* const [showPassword, setShowPassword] = useState(false);
* const [loading, setLoading] = useState(false);
* // Handle input change
* const handleChange = (e) => {
* const { name, value } = e.target;
* setFormData((prev) => ({ ...prev, [name]: value }));
* };
* // Validate email format
* const validateEmail = (email) => /\S+@\S+\.\S+/.test(email);
* // Handle form submit
* const handleSubmit = async (e) => {
* e.preventDefault();
* setLoading(true);
* // Trim & lowercase email, trim password
* const email = String(formData.email || '').trim().toLowerCase();
* const password = String(formData.password || '').trim();
* if (!email || !password) {
* alert('Vui lòng nhập email và mật khẩu.');
* setLoading(false);
* return;
* }
* if (!validateEmail(email)) {
* alert('Email không hợp lệ.');
* setLoading(false);
* return;
* }
* try {
* const res = await authService.login({ email, password });
* if (!res?.success) throw new Error(res.message || 'Đăng nhập thất bại');
* const { user } = res;
* if (!user) throw new Error('Không nhận được thông tin người dùng.');
* // Điều hướng theo role
* switch (user.role) {
* case 'admin':
* navigate('/admin');
* break;
* case 'doctor':
* navigate('/doctor');
* break;
* case 'patient':
* navigate('/patient');
* break;
* default:
* navigate('/');
* }
* } catch (err) {
* console.error('[Login] Error:', err);
* alert(err.message);
* } finally {
* setLoading(false);
* }
* };
* return (
* <div className="auth-container">
* <div className="auth-left">
* <h1>Telehealth System</h1>
* <p>Hệ thống Đặt Lịch & Theo Dõi Khám Bệnh Từ Xa</p>
* </div>
* <div className="auth-right">
* <form onSubmit={handleSubmit} className="auth-form">
* <h2>Đăng nhập</h2>
* <label>Email</label>
* <input
* type="email"
* name="email"
* value={formData.email}
* onChange={handleChange}
* placeholder="email@example.com"
* required
* />
* <label>Mật khẩu</label>
* <div className="password-wrapper">
* <input
* type={showPassword ? 'text' : 'password'}
* name="password"
* value={formData.password}
* onChange={handleChange}
* placeholder="Tối thiểu 6 ký tự"
* required
* />
* <button
* type="button"
* className="toggle-password"
* onClick={() => setShowPassword((prev) => !prev)}
* aria-label="Hiển thị mật khẩu"
* >
* {showPassword ? 'Ẩn' : 'Hiện'}
* </button>
* </div>
* <button type="submit" className="auth-btn" disabled={loading}>
* {loading ? 'Đang đăng nhập...' : 'Đăng nhập'}
* </button>
* <p className="auth-footer">
* Chưa có tài khoản?{' '}
* <span onClick={() => navigate('/register')}>Đăng ký ngay</span>
* </p>
* </form>
* </div>
* </div>
* );
* };
* export default LoginPage;
* **AppointmentForm.jsx (Trang đặt lịch bệnh nhân)**
* import React, { useState, useEffect, useCallback, useRef } from 'react';
* import bookingService from '../../services/bookingService';
* import './styles/AppointmentForm.css';
* import dayjs from 'dayjs';
* import { formatVN } from '../../utils/timezone';
* const AppointmentForm = ({ currentUser }) => {
* const [locations, setLocations] = useState([]);
* const [specialties, setSpecialties] = useState([]);
* const [doctors, setDoctors] = useState([]);
* const [availableTimes, setAvailableTimes] = useState([]);
* const [loadingSlots, setLoadingSlots] = useState(false);
* const [bookingData, setBookingData] = useState({
* location: '',
* specialty: '',
* doctor: '',
* date: '',
* time: '',
* patient: { reason: '' },
* });
* const [loadingSubmit, setLoadingSubmit] = useState(false);
* const [otpStage, setOtpStage] = useState(false);
* const [otpInput, setOtpInput] = useState('');
* const [createdAppointmentId, setCreatedAppointmentId] = useState(null);
* const [otpTimer, setOtpTimer] = useState(0);
* const [resendCooldown, setResendCooldown] = useState(0);
* const specialtiesCache = useRef({});
* const doctorsCache = useRef({});
* const today = new Date().toISOString().split('T')[0];
* const showToast = useCallback(msg => alert(msg), []);
* // Load Locations
* useEffect(() => {
* (async () => {
* try {
* const res = await bookingService.fetchLocations();
* setLocations(Array.isArray(res) ? res : []);
* } catch (err) {
* console.error(err);
* showToast('Lỗi load cơ sở y tế');
* }
* })();
* }, [showToast]);
* // Load Specialties
* const loadSpecialties = useCallback(async (locationId) => {
* if (!locationId) return setSpecialties([]);
* if (specialtiesCache.current[locationId]) {
* setSpecialties(specialtiesCache.current[locationId]);
* return;
* }
* try {
* const res = await bookingService.fetchSpecialties(locationId);
* const list = Array.isArray(res) ? res : [];
* setSpecialties(list);
* specialtiesCache.current[locationId] = list;
* } catch (err) {
* console.error(err);
* showToast('Lỗi load chuyên khoa');
* setSpecialties([]);
* }
* }, [showToast]);
* useEffect(() => {
* setSpecialties([]);
* setDoctors([]);
* setBookingData(prev => ({ ...prev, specialty: '', doctor: '' }));
* if (bookingData.location) loadSpecialties(bookingData.location);
* }, [bookingData.location, loadSpecialties]);
* // Load Doctors
* const loadDoctors = useCallback(async () => {
* if (!bookingData.location || !bookingData.specialty) return setDoctors([]);
* const key = `${bookingData.location}\_${bookingData.specialty}`;
* if (doctorsCache.current[key]) {
* setDoctors(doctorsCache.current[key]);
* return;
* }
* try {
* const res = await bookingService.fetchDoctors(bookingData.location, bookingData.specialty);
* const list = Array.isArray(res) ? res : [];
* setDoctors(list);
* doctorsCache.current[key] = list;
* } catch (err) {
* console.error(err);
* showToast('Lỗi load bác sĩ');
* setDoctors([]);
* }
* }, [bookingData.location, bookingData.specialty, showToast]);
* useEffect(() => {
* setDoctors([]);
* setBookingData(prev => ({ ...prev, doctor: '' }));
* loadDoctors();
* }, [bookingData.location, bookingData.specialty, loadDoctors]);
* // Load Available Slots
* const loadAvailableSlots = useCallback(async () => {
* if (!bookingData.doctor || !bookingData.date) {
* setAvailableTimes([]);
* return;
* }
* setLoadingSlots(true);
* try {
* const res = await bookingService.fetchAvailableSlots(bookingData.doctor, bookingData.date);
* const slots = Array.isArray(res) ? res : (res?.data?.availableSlots || []);
* const normalized = slots.map(slot => {
* const timeStr = typeof slot === 'string' ? slot : slot.time || '';
* const datetimeVN = slot.datetimeVN ? formatVN(slot.datetimeVN) : formatVN(timeStr);
* const displayTime = dayjs(datetimeVN).format('HH:mm');
* return { time: timeStr, displayTime, datetimeVN };
* });
* setAvailableTimes(normalized);
* } catch (err) {
* console.error(err);
* showToast('Không thể tải danh sách giờ khám.');
* setAvailableTimes([]);
* } finally {
* setLoadingSlots(false);
* }
* }, [bookingData.doctor, bookingData.date, showToast]);
* useEffect(() => {
* setAvailableTimes([]);
* setBookingData(prev => ({ ...prev, time: '' }));
* loadAvailableSlots();
* }, [bookingData.doctor, bookingData.date, loadAvailableSlots]);
* // OTP timers
* useEffect(() => {
* if (otpTimer <= 0) return;
* const interval = setInterval(() => setOtpTimer(prev => Math.max(prev - 1, 0)), 1000);
* return () => clearInterval(interval);
* }, [otpTimer]);
* useEffect(() => {
* if (resendCooldown <= 0) return;
* const interval = setInterval(() => setResendCooldown(prev => Math.max(prev - 1, 0)), 1000);
* return () => clearInterval(interval);
* }, [resendCooldown]);
* // Handlers
* const handlePatientChange = e => {
* const { name, value } = e.target;
* setBookingData(prev => ({ ...prev, patient: { ...prev.patient, [name]: value } }));
* };
* const validateForm = () => {
* const { location, specialty, doctor, date, time, patient } = bookingData;
* if (!location || !specialty || !doctor || !date || !time) {
* showToast('Vui lòng bổ sung đầy đủ thông tin.');
* return false;
* }
* if (!patient.reason) {
* showToast('Vui lòng nhập lý do khám.');
* return false;
* }
* if (!availableTimes || availableTimes.length === 0) {
* showToast('Không còn slot trống cho ngày này.');
* return false;
* }
* return true;
* };
* const handleSubmit = async e => {
* e.preventDefault();
* if (!validateForm()) return;
* setLoadingSubmit(true);
* try {
* const payload = {
* locationId: bookingData.location,
* specialtyId: bookingData.specialty,
* doctorId: bookingData.doctor,
* date: bookingData.date,
* time: bookingData.time, // gửi string "HH:mm"
* reason: bookingData.patient.reason
* };
* const res = await bookingService.createAppointment(payload);
* const appointmentId = res?.data?.appointment?.\_id;
* if (!appointmentId) throw new Error('Không thể đặt lịch ở quá khứ ');
* setCreatedAppointmentId(appointmentId);
* localStorage.setItem('tempAppointmentId', appointmentId);
* setOtpStage(true);
* setOtpTimer(300);
* setResendCooldown(30);
* showToast('Đặt lịch thành công! OTP đã gửi vui lòng kiểm tra email.');
* } catch (err) {
* console.error(err);
* showToast(err.message || 'Đặt lịch thất bại.');
* } finally {
* setLoadingSubmit(false);
* }
* };
* const handleVerifyOtp = async () => {
* if (!otpInput) return showToast('Vui lòng nhập OTP');
* const appointmentId = createdAppointmentId || localStorage.getItem('tempAppointmentId');
* if (!appointmentId) return showToast('Lỗi: Không tìm thấy thông tin lịch hẹn');
* try {
* await bookingService.verifyOtp(appointmentId, otpInput);
* showToast('Xác thực OTP thành công');
* localStorage.removeItem('tempAppointmentId');
* resetForm();
* } catch (err) {
* console.error(err);
* showToast(err.message || 'Xác thực OTP thất bại');
* }
* };
* const handleResendOtp = async () => {
* const appointmentId = createdAppointmentId || localStorage.getItem('tempAppointmentId');
* if (!appointmentId || resendCooldown > 0) return;
* setResendCooldown(30);
* setOtpTimer(300);
* try {
* await bookingService.resendOtp(appointmentId);
* showToast('OTP đã gửi lại');
* } catch (err) {
* console.error(err);
* showToast(err.message || 'Gửi lại OTP thất bại');
* }
* };
* const resetForm = () => {
* setOtpStage(false);
* setOtpInput('');
* setCreatedAppointmentId(null);
* localStorage.removeItem('tempAppointmentId');
* setOtpTimer(0);
* setResendCooldown(0);
* setBookingData({ location:'', specialty:'', doctor:'', date:'', time:'', patient:{reason:''} });
* setSpecialties([]);
* setDoctors([]);
* setAvailableTimes([]);
* specialtiesCache.current = {};
* doctorsCache.current = {};
* };
* const formatTimer = sec => {
* const m = Math.floor(sec / 60);
* const s = sec % 60;
* return `${m}:${s.toString().padStart(2,'0')}`;
* };
* return (
* <div className="appointment-form-container">
* <h2 className="form-title">Thông tin lịch hẹn</h2>
* {!otpStage ? (
* <form className="appointment-form-grid" onSubmit={handleSubmit}>
* <div className="form-section card">
* <h3 className="section-title">Thông tin lịch khám</h3>
* <div className="form-group">
* <label>Cơ sở y tế <span className="required">\*</span></label>
* <select value={bookingData.location} onChange={e => setBookingData(prev => ({ ...prev, location: e.target.value }))}>
* <option value="">Chọn cơ sở y tế</option>
* {locations.map(loc => <option key={loc.\_id} value={loc.\_id}>{loc.name}</option>)}
* </select>
* </div>
* <div className="form-group">
* <label>Chuyên khoa <span className="required">\*</span></label>
* <select value={bookingData.specialty} onChange={e => setBookingData(prev => ({ ...prev, specialty: e.target.value }))} disabled={!bookingData.location}>
* <option value="">Chọn chuyên khoa</option>
* {specialties.map(s => <option key={s.\_id} value={s.\_id}>{s.name}</option>)}
* </select>
* </div>
* <div className="form-group">
* <label>Bác sĩ <span className="required">\*</span></label>
* <select value={bookingData.doctor} onChange={e => setBookingData(prev => ({ ...prev, doctor: e.target.value }))} disabled={!bookingData.specialty}>
* <option value="">Chọn bác sĩ</option>
* {doctors.map(d => <option key={d.\_id} value={d.\_id}>{d.user?.fullName || 'Không xác định'}</option>)}
* </select>
* </div>
* <div className="form-group">
* <label>Ngày khám <span className="required">\*</span></label>
* <input type="date" value={bookingData.date} min={today} onChange={e => setBookingData(prev => ({ ...prev, date: e.target.value }))} disabled={!bookingData.doctor} />
* </div>
* <div className="form-group">
* <label>Chọn giờ <span className="required">\*</span></label>
* {loadingSlots && <p className="loading-text">⏳ Đang tải khung giờ...</p>}
* <select value={bookingData.time} onChange={e => setBookingData(prev => ({ ...prev, time: e.target.value }))} disabled={!bookingData.date || availableTimes.length === 0 || loadingSlots}>
* <option value="">{loadingSlots ? 'Đang tải...' : availableTimes.length === 0 ? 'Không có khung giờ trống' : 'Chọn giờ khám'}</option>
* {availableTimes.map((slot, i) => (
* <option key={i} value={slot.time}>
* {slot.displayTime}
* </option>
* ))}
* </select>
* </div>
* </div>
* <div className="form-section card">
* <h3 className="section-title">Triệu chứng / Lý do khám</h3>
* <div className="form-group">
* <textarea name="reason" placeholder="Vui lòng nhập triệu chứng..." value={bookingData.patient.reason} onChange={handlePatientChange} rows={4} />
* </div>
* </div>
* <button type="submit" className="btn-submit" disabled={loadingSubmit || loadingSlots || availableTimes.length === 0}>
* {loadingSubmit ? 'Đang gửi...' : 'Đặt lịch'}
* </button>
* </form>
* ) : (
* <div className="otp-verification card">
* <h3 className="section-title">Nhập mã OTP để xác nhận lịch hẹn</h3>
* <input type="text" value={otpInput} onChange={e => setOtpInput(e.target.value)} placeholder="Mã OTP" maxLength={6} />
* <div className="otp-actions">
* <button type="button" onClick={handleVerifyOtp}>Xác nhận OTP</button>
* <button type="button" onClick={handleResendOtp} disabled={resendCooldown > 0} className={resendCooldown > 0 ? 'disabled' : ''}>
* {resendCooldown > 0 ? `Gửi lại sau ${resendCooldown}s` : 'Gửi lại OTP'}
* </button>
* </div>
* {otpTimer > 0 && <p className="otp-timer">OTP còn hiệu lực: {formatTimer(otpTimer)}</p>}
* </div>
* )}
* </div>
* );
* };
* export default AppointmentForm;

***Ghi chú: Chi tiết đầy đủ trong GitHub và Source code đính kèm***

**MongoDB**

**Lưu trữ dữ liệu bao gồm các collections như users, patients, doctors, appointments,…**



**Phụ lục C – Seed Script & Data Demo**

**C.1 File script seed.js**

const mongoose = require('mongoose');

const dayjs = require('dayjs');

const utc = require('dayjs/plugin/utc');

const timezone = require('dayjs/plugin/timezone');

const dotenv = require('dotenv');

dotenv.config();

dayjs.extend(utc);

dayjs.extend(timezone);

const { toVN } = require('../utils/timezone');

const User = require('../models/User');

const Doctor = require('../models/Doctor');

const Patient = require('../models/Patient');

const Location = require('../models/Location');

const Specialty = require('../models/Specialty');

const Schedule = require('../models/Schedule');

const Appointment = require('../models/Appointment');

const MedicalRecord = require('../models/MedicalRecord');

const MONGO\_URI = process.env.MONGODB\_URI || 'mongodb://127.0.0.1:27017/telehealth';

mongoose.connect(MONGO\_URI)

  .then(() => console.log('>>> MongoDB connected'))

  .catch(err => {

    console.error('>>> MongoDB connection error:', err);

    process.exit(1);

  });

// Hàm lấy 7 ngày liên tiếp

const getNextDates = (numDays) => {

  const dates = [];

  const today = dayjs();

  for (let i = 0; i < numDays; i++) {

    dates.push(today.add(i, 'day').format('YYYY-MM-DD'));

  }

  return dates;

};

const seedData = async () => {

  try {

    console.log('>>> Clearing old data...');

    await Appointment.deleteMany({});

    await Schedule.deleteMany({});

    await Doctor.deleteMany({});

    await Patient.deleteMany({});

    await User.deleteMany({});

    await Location.deleteMany({});

    await Specialty.deleteMany({});

    // Tạo 3 Locations

    const locationsData = [

      { name: 'Bệnh viện A', address: '123 Đường A' },

      { name: 'Bệnh viện B', address: '456 Đường B' },

      { name: 'Bệnh viện C', address: '789 Đường C' }

    ];

    const locations = await Location.insertMany(locationsData);

    console.log(`>>> ${locations.length} Locations created`);

    // Tạo 5 Specialties mỗi location

    const specialtyNames = ['Nội', 'Ngoại', 'Răng', 'Mắt', 'Tai Mũi Họng'];

    const specialties = [];

    for (const loc of locations) {

      for (const name of specialtyNames) {

        specialties.push({ name, location: loc.\_id });

      }

    }

    const createdSpecialties = await Specialty.insertMany(specialties);

    console.log(`>>> ${createdSpecialties.length} specialties created`);

    // Tạo 10 Doctor Users

    const doctorUsers = [];

    for (let i = 0; i < 10; i++) {

      const randomSpecialty = createdSpecialties[Math.floor(Math.random() \* createdSpecialties.length)];

      const locId = randomSpecialty.location;

      const email = `doctor${i + 1}@demo.com`;

      let existingUser = await User.findOne({ email });

      if (!existingUser) {

        const user = await User.create({

          fullName: `Doctor ${i + 1}`,

          email,

          password: '123456', // hook sẽ hash

          role: 'doctor',

          specialty: randomSpecialty.\_id,

          location: locId

        });

        doctorUsers.push(user);

      } else {

        doctorUsers.push(existingUser);

      }

    }

    console.log(`>>> ${doctorUsers.length} doctor users created`);

    // Lấy tất cả Doctor documents (hook tự tạo)

    const doctorDocs = await Doctor.find({});

    if (doctorDocs.length === 0) throw new Error('No Doctor documents found after creating users');

    // Tạo 20 Patient Users

    const patientDocs = [];

    for (let i = 1; i <= 20; i++) {

      const email = `patient${i}@demo.com`;

      let user = await User.findOne({ email });

      if (!user) {

        user = await User.create({

          fullName: `Patient ${i}`,

          email,

          password: '123456',

          role: 'patient'

        });

      }

      let patient = await Patient.findOne({ user: user.\_id });

      if (!patient) {

        patient = await Patient.create({

          user: user.\_id,

          address: `${i} Phố Demo, TP.HCM`,

          bio: '',

          medicalHistory: ''

        });

      }

      patientDocs.push(patient);

    }

    console.log(`>>> ${patientDocs.length} Patients created`);

    // Schedules

    const scheduleDocs = [];

    for (const doctor of doctorDocs) {

      const dates = getNextDates(7);

      for (const date of dates) {

        const dayOfWeek = dayjs(date).day(); // 0 = CN

        let slots = [];

        if (dayOfWeek !== 0) { // CN nghỉ

          for (let h = 8; h < 11; h++) {

            slots.push({ time: `${h.toString().padStart(2,'0')}:00` });

            slots.push({ time: `${h.toString().padStart(2,'0')}:30` });

          }

          for (let h = 13; h < 17; h++) {

            slots.push({ time: `${h.toString().padStart(2,'0')}:00` });

            slots.push({ time: `${h.toString().padStart(2,'0')}:30` });

          }

        }

        const schedule = await Schedule.create({

          doctorId: doctor.\_id,

          date,

          slots

        });

        scheduleDocs.push(schedule);

      }

    }

    console.log(`>>> ${scheduleDocs.length} Schedules created`);

    // Tạo 50 Appointments random

    const statuses = ['pending', 'confirmed', 'cancelled', 'completed', 'expired'];

    const appointmentDocs = [];

    for (let i = 0; i < 50; i++) {

      const patient = patientDocs[Math.floor(Math.random() \* patientDocs.length)];

      const doctor = doctorDocs[Math.floor(Math.random() \* doctorDocs.length)];

      // Lọc schedule còn slot trống của bác sĩ

      const doctorSchedules = scheduleDocs.filter(

        s => s.doctorId.equals(doctor.\_id) && s.slots.some(slot => !slot.isBooked)

      );

      if (doctorSchedules.length === 0) continue;

      const schedule = doctorSchedules[Math.floor(Math.random() \* doctorSchedules.length)];

      const freeSlots = schedule.slots.filter(s => !s.isBooked);

      const slot = freeSlots[Math.floor(Math.random() \* freeSlots.length)];

      // Đánh dấu slot đã book và lưu schedule

      slot.isBooked = true;

      await schedule.save();

      // Tạo datetime luôn trong tương lai

      let datetimeRaw = dayjs(`${schedule.date}T${slot.time}:00`);

      let datetime = datetimeRaw.isBefore(dayjs())

        ? dayjs().add(1, 'day').hour(datetimeRaw.hour()).minute(datetimeRaw.minute())

        : datetimeRaw;

      datetime = datetime.toDate();

      const appointment = await Appointment.create({

        location: doctor.location,

        specialty: doctor.specialty,

        doctor: doctor.\_id,

        patient: patient.\_id,

        datetime,

        date: dayjs(datetime).format('YYYY-MM-DD'),

        time: dayjs(datetime).format('HH:mm'),

        reason: `Khám bệnh test ${i + 1}`,

        status: statuses[Math.floor(Math.random() \* statuses.length)],

        isVerified: Math.random() < 0.8,

        workScheduleId: schedule.\_id

      });

      appointmentDocs.push(appointment);

    }

    console.log(`>>> ${appointmentDocs.length} Appointments created`);

    // Tạo MedicalRecords appointments có trạng thái completed

    const completedAppointments = appointmentDocs.filter(a => a.status === 'completed');

    for (const appt of completedAppointments) {

      const doctorData = await Doctor.findById(appt.doctor).populate('user');

      const patientData = await Patient.findById(appt.patient).populate('user');

      const specialtyData = await Specialty.findById(appt.specialty);

      const locationData = await Location.findById(appt.location);

      await MedicalRecord.create({

        appointment: appt.\_id,

        doctor: appt.doctor,

        patient: appt.patient,

        doctorSnapshot: {

          fullName: doctorData.user.fullName,

          specialty: specialtyData.\_id.toString(),

          specialtyName: specialtyData.name,

          location: locationData.\_id.toString(),

          locationName: locationData.name

        },

        patientSnapshot: {

          fullName: patientData.user.fullName,

          birthYear: patientData.user.birthYear,

          address: patientData.address,

          phone: patientData.user.phone

        },

        date: appt.datetime,

        height: 170,

        weight: 70,

        bp: '120/80',

        pulse: 70,

        bmi: 24.2,

        symptoms: 'Đau đầu, chóng mặt',

        diagnosis: 'Cảm cúm',

        notes: 'Test medical record',

        prescriptions: [

          { name: 'Paracetamol', dose: '500mg', quantity: 10, note: '2 viên mỗi lần' }

        ],

        conclusion: 'Bình thường',

        careAdvice: 'Uống nhiều nước, nghỉ ngơi'

      });

    }

    console.log(`>>> ${completedAppointments.length} MedicalRecords created`);

    console.log('>>> Seed data completed!');

    process.exit(0);

  } catch (err) {

    console.error('>>> Error seeding data:', err);

    process.exit(1);

  }

};

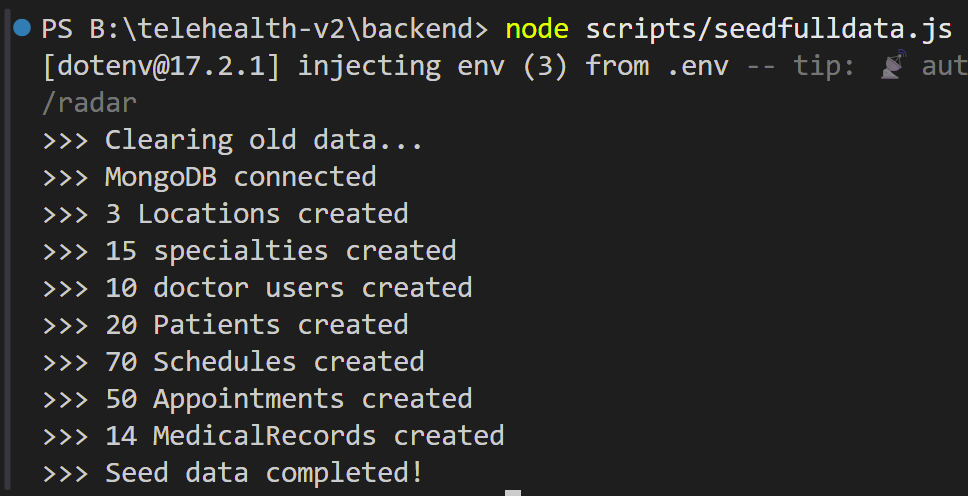
seedData();

**C.2 Dữ liệu mẫu**

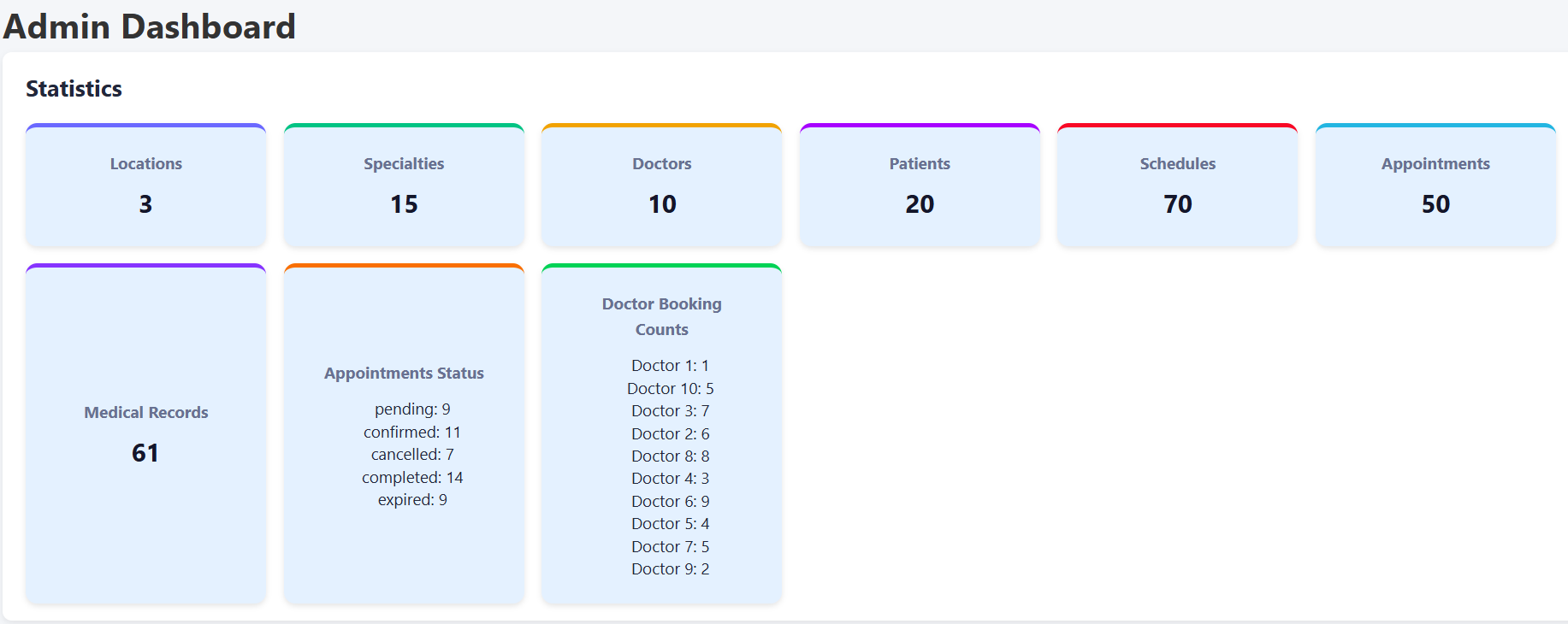
* 1 Admin: admin@telehealth.com
* 5 Bác sĩ: thuộc các chuyên khoa (Nội, Ngoại, Nhi, Tim mạch, Da liễu)
* 10 Bệnh nhân: tạo ngẫu nhiên
* 20 Lịch hẹn: rải trong tuần hiện tại

Bảng 3 Kết quả seed dữ liệu test

| Admin | Bệnh nhân | Bác sĩ | Số lịch hẹn | Đã xác nhận | Chờ xác thực |
| --- | --- | --- | --- | --- | --- |
| 1 | 20 | 10 | 50 | 7 | 3 |



**Thống kê từ AdminDashboard**



**Phụ lục D – Test Case Chi Tiết**

**D.1 Patient Flow**

Bảng 4 TestCase Patient - Doctor - Admin Flow

| Test ID | Mô tả kịch bản | Input / Thao tác | Kết quả mong đợi | Kết quả thực tế | Đạt/Không đạt |
| --- | --- | --- | --- | --- | --- |
| P01 | Đăng ký tài khoản | Nhập thông tin (username, email, password) → Submit | Tài khoản mới được đăng ký, nhận thông báo thành công | Đăng ký tài khoản thông báo thành công | Đạt |
| P02 | Đăng nhập hệ thống | Nhập email & password hợp lệ → Submit | Đăng nhập thành công, chuyển về dashboard bệnh nhân | Đăng nhập tài khoản thông báo thành công | Đạt |
| P03 | Xem danh sách cơ sở y tế | Chọn menu “Đặt lịch”, click chọn ô “Cơ sở y tế” | Hiển thị danh sách cơ sở y tế từ DB | Hiển thị đầy đù danh sách cơ sở y tế | Đạt |
| P04 | Xem danh sách chuyên khoa | Click vào một cơ sở y tế => hiển thị chuyên khoa | Danh sách chuyên khoa thuộc cơ sở y tế được hiển thị đúng | Hiển thị chuyên khoa thuộc cơ sở y tế đã chọn | Đạt |
| P05 | Chọn bác sĩ theo chuyên khoa | Click vào một chuyên khoa => hiển thị bác sĩ | Danh sách bác sĩ thuộc chuyên khoa được hiển thị đúng | Hiển thị bác sĩ thuộc chuyên khoa đã chọn | Đạt |
| P06 | Đặt lịch khám | Click chọn bác sĩ => Chọn ngày + giờ trống lịch + nhập triệu chứng / lý do khám => Đặt lịch | Lịch hẹn được tạo, hiển thị trạng thái “Pending” và có thông báo yêu cầu xác thực | Đặt lịch hẹn có thông báo yêu cầu xác thực OTP | Đạt |
| P07 | Xác thực | Đặt lịch => nhập OTP 6 số => xác nhận | Thông báo đặt lịch thành công | Xác thực OTP và có thông báo đặt lịch thành công | Đạt |
| P08 | Xem lịch sử cuộc hẹn | Vào menu “Lịch sử” | Hiển thị danh sách lịch sử đặt hẹn | Hiển thị danh sách lịch sử đặt hẹn, theo dõi trạng thái cuộc hẹn | Đạt |

**D.2 Doctor Flow**

| **Test ID** | **Mô tả kịch bản** | **Input / Thao tác** | **Kết quả mong đợi** | **Kết quả thực tế** | **Đạt/Không đạt** |
| --- | --- | --- | --- | --- | --- |
| P01 | Đăng ký tài khoản | Nhập thông tin (username, email, password, chuyên khoa, cơ sở y tế) → Submit | Tài khoản mới được đăng ký, nhận thông báo thành công | Đăng ký tài khoản thông báo thành công | Đạt |
| P02 | Đăng nhập hệ thống | Nhập email & password hợp lệ → Submit | Đăng nhập thành công, chuyển về dashboard bác sĩ | Đăng nhập tài khoản thông báo thành công | Đạt |
| D03 | Xác nhận lịch hẹn | Vào menu “Lịch làm” nhấn nút xác nhận cuộc hẹn | Trạng thái lịch chuyển từ “Pending” → “Confirmed” | Chưa triển khai | Không đạt |
| D04 | Xem lịch làm việc ngày / tuần | Vào menu “Trang chủ / lịch làm” | Hiển thị danh sách lịch hẹn trong ngày (theo giờ hành chính) dạng grid-slot | Trang chủ hiển thị nhanh lịch hẹn trong ngày, chỉ hiển thị lịch có trạng thái đã xác nhận (confirmed) | Đạt |
| D05 | Xem chi tiết lịch hẹn | Click chọn cuộc hẹn trực tiếp trong grid-slot để xem chi tiết lịch hẹn | Hiển thị thông tin chi tiết lịch hẹn + thông tin liên hệ của bệnh nhân | Hiển thị đúng chi tiết cuộc hẹn + thông tin bệnh nhân | Đạt |
| D06 | Tạo hồ sơ khám bệnh | Click nút tạo phiếu / hồ sơ khám bệnh → Nhập thông tin chi tiêt bệnh nhân, chỉ số cơ thể, chẩn đoán & đơn thuốc → Nhân “Lưu” | Hồ sơ khám bệnh được tạo thành công có thông báo | Hồ sơ khám bệnh được tạo, trạng thái cuộc hẹn chuyển sang completed | Đạt |
| D07 | Xem lại hồ sơ khám bệnh | Vào menu “Quản lý bệnh nhân / hồ sơ bệnh án” => Click chọn một bệnh nhân => Click xem hồ sơ | Danh sách hồ sơ y tế đã tạo hiển thị đúng | Hiển thị đúng danh sách các hồ sơ bệnh án của bệnh nhân | Đạt |

**D.3 Admin Flow**

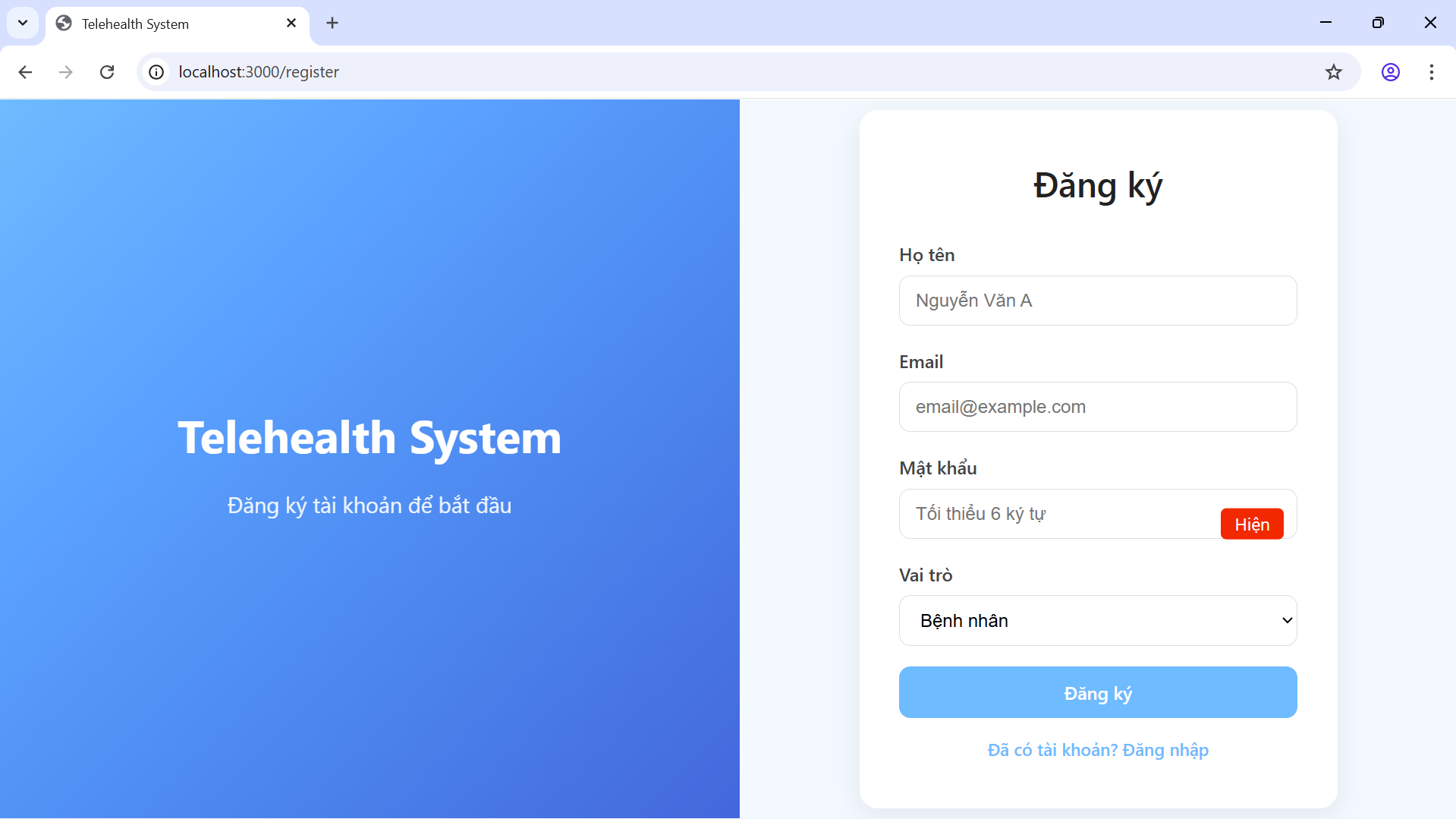
| **Test ID** | **Mô tả kịch bản** | **Input / Thao tác** | **Kết quả mong đợi** | **Kết quả thực tế** | **Đạt / Không đạt** |
| --- | --- | --- | --- | --- | --- |
| P01 | Tạo mới user | Tại giao diện AdminDashboard mục Users => Click nút “Thêm user” => |  |  |  |
|  |  | P01.1 Chọn role patient => Nhập fullName, email đăng ký, mật khẩu => Click nút “Thêm” | Thêm patient user thành công vào dữ liệu DB | Thêm patient user thành công cập nhật danh sách User | Đạt |
|  |  | P02.2 Chọn role doctor => Nhập fullName, email đăng ký, mật khẩu => Chọn một cơ sở y tế load chuyên khoa thuôc cơ sở y tế đó => Chọn một chuyên khoa => Click nút “Thêm” | Thêm doctor user thành công vào dữ liệu DB | Thêm doctor user thành công cập nhật danh sách User – Doctors | Đạt |
| P02 | Sửa thông tin user | Tại giao diện AdminDashboard mục Users => Chọn một user click nút “Sửa” => cho phép admin sửa thông tin gồm tên, email, role, chuyên khoa – cơ sở (đối với doctor user) => Click nút “Cập nhật” | Sửa / cập nhật thông tin user thành công dữ liệu DB | Sửa / cập nhật thông tin user thành công | Đạt |
| P03 | Xóa user | Tại giao diện AdminDashboard mục User => chọn một user click nút “Xóa” => hệ thống xác nhận xóa => click xóa / hủy => click xóa | Xóa thành công user khỏi dữ liệu DB | Xóa user tức thời khỏi danh sách | Đạt |
| P04 | Thêm location | Tại giao diện AdminDashboard mục Location => click nút “Thêm location” => nhập tên cơ sở y tế, địa chỉ => click nút “Thêm” | Thêm cơ sở y tế mới vào dữ liệu DB | Thêm thành công cơ sở y tế mới | Đạt |
| P05 | Sửa / Cập nhật location | Tại giao diện AdminDashboard mục Location => click nút “Sửa” => cho phép sửa thông tin tên cơ sở y tế, địa chỉ | Sửa / cập nhật thông tin cơ sở y tế dữ liệu DB | Sửa / cập nhật thành công thông tin cơ sở y tế | Đạt |
| P06 | Thêm / Cập nhật / Xóa specialty | Tại giao diện AdminDashboard mục Location => chọn một cơ sở y tế Click nút “Sửa” tại giao diện Modal “Sửa location” => Nhập tên chuyên khoa => Click nút thêm để thêm special / nút “x” để xóa special => click nút “Cập nhật” P06.1 Special đó có doctor không được phép xóa | Thêm / cập nhật chuyên khoa thành công dữ liệu DB | Thêm / cập nhật thành công chuyên khoa thuộc cơ sở y tế | Đạt |
| P07 | Xóa Location | Tại giao diện AdminDashboard mục Location => chọn một cơ sở y tế click nút “Xóa” => hệ thống xác nhận xóa / hủy => Click nút “Xóa / hủy” P07 Location đó có gắn với specialty – doctor không được phép xóa | Xóa thành công Location khỏi dữ liệu DB | Xóa Location thành công khỏi danh sách | Đạt |

**Phụ lục E – Giao diện (Screenshots)**

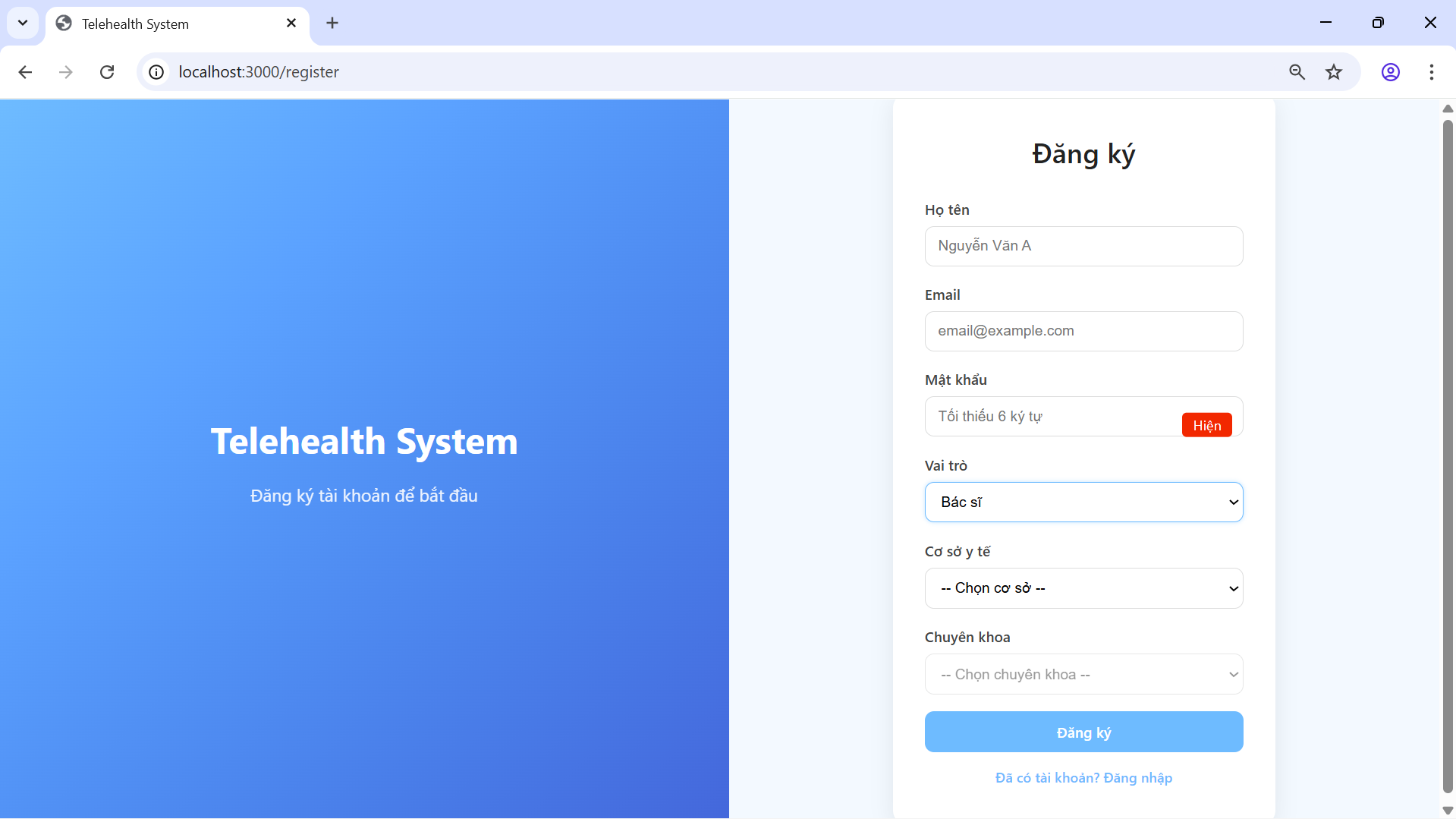
**E.1 Đăng kỳ / Đăng nhập**

**E.1.1 RegisterPage**

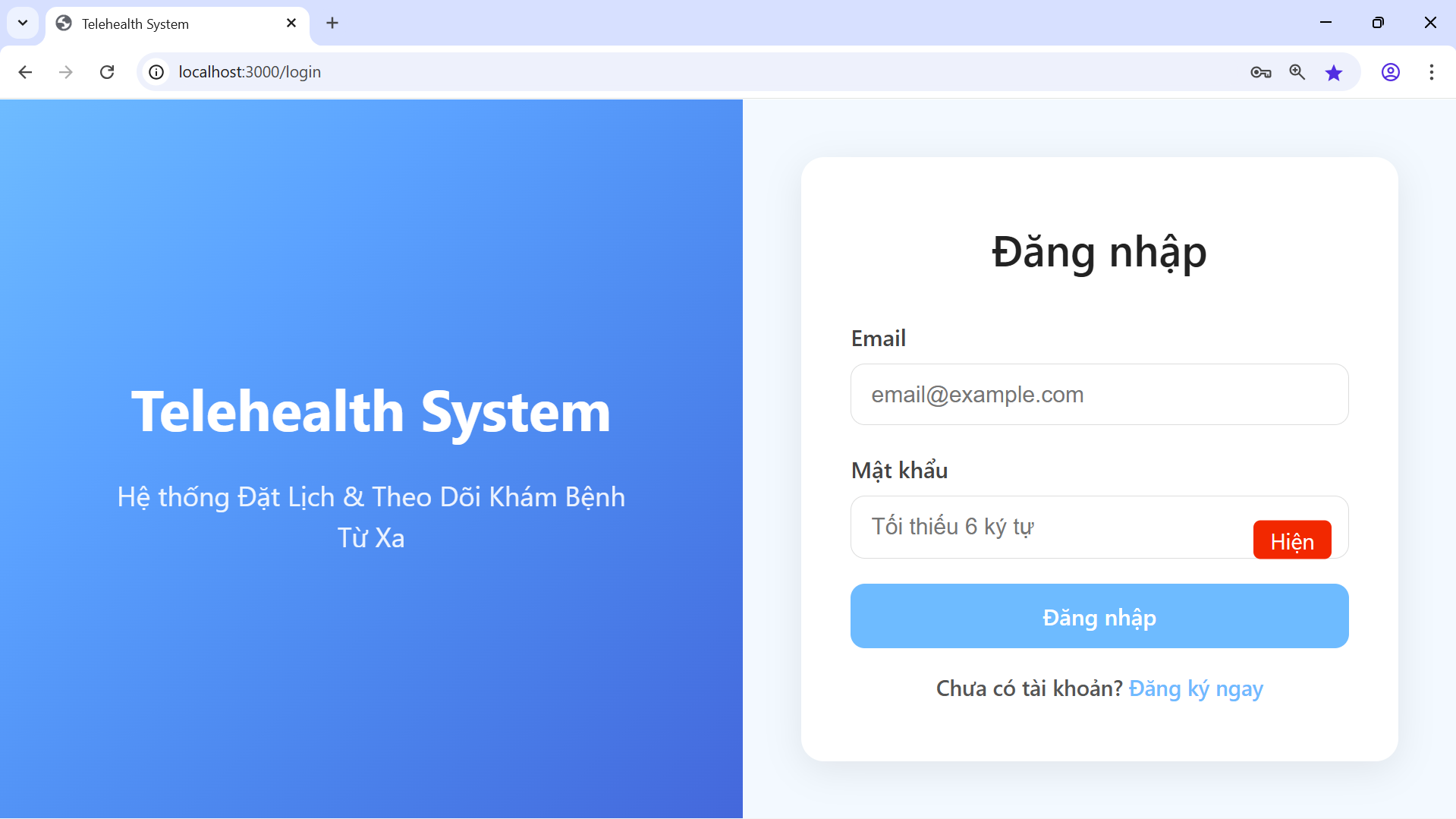
**Đăng ký vai trò bệnh nhân**

****

**Đăng ký vai trò bác sĩ**

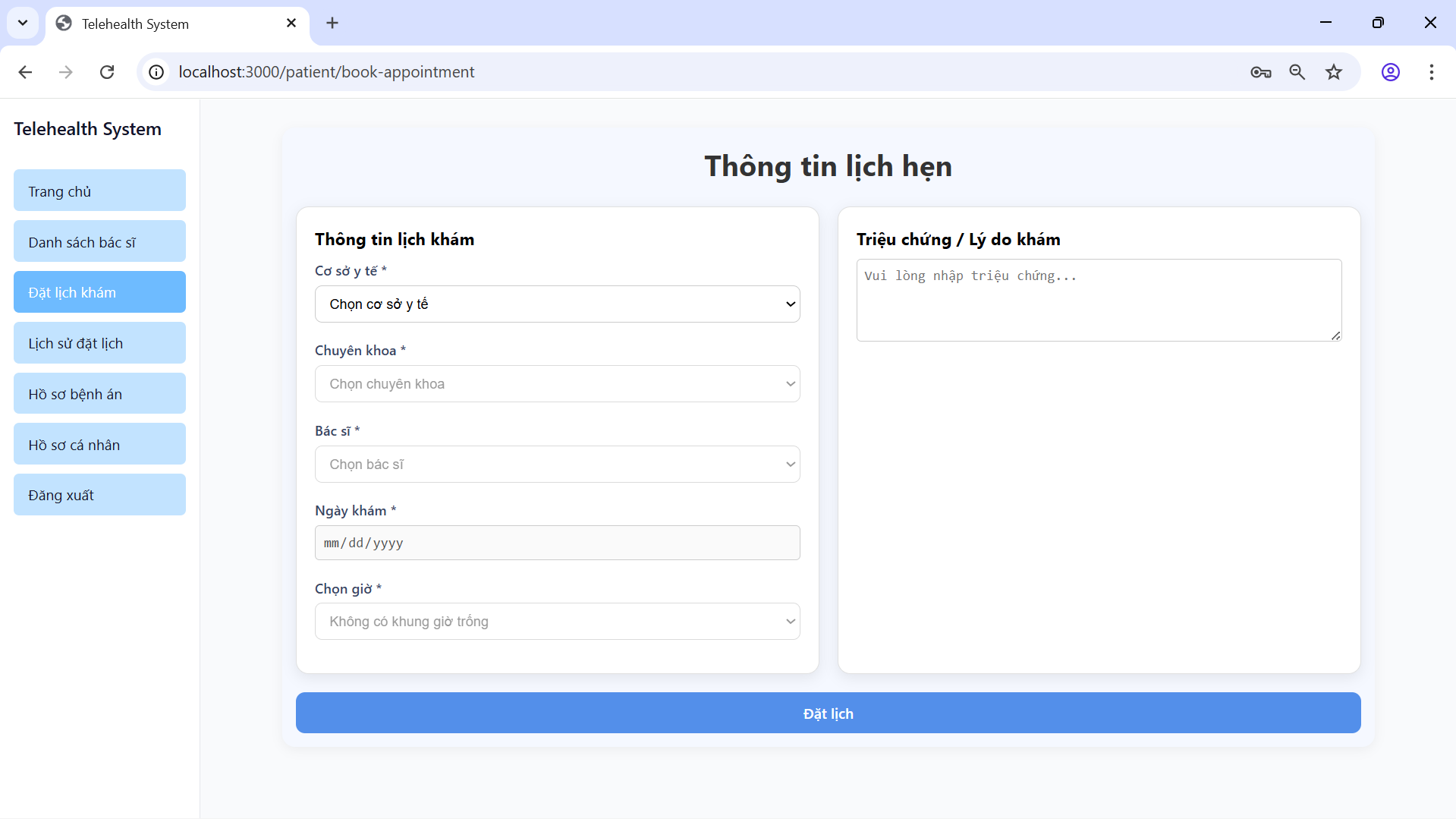
****

**E.1.2 LoginPage**

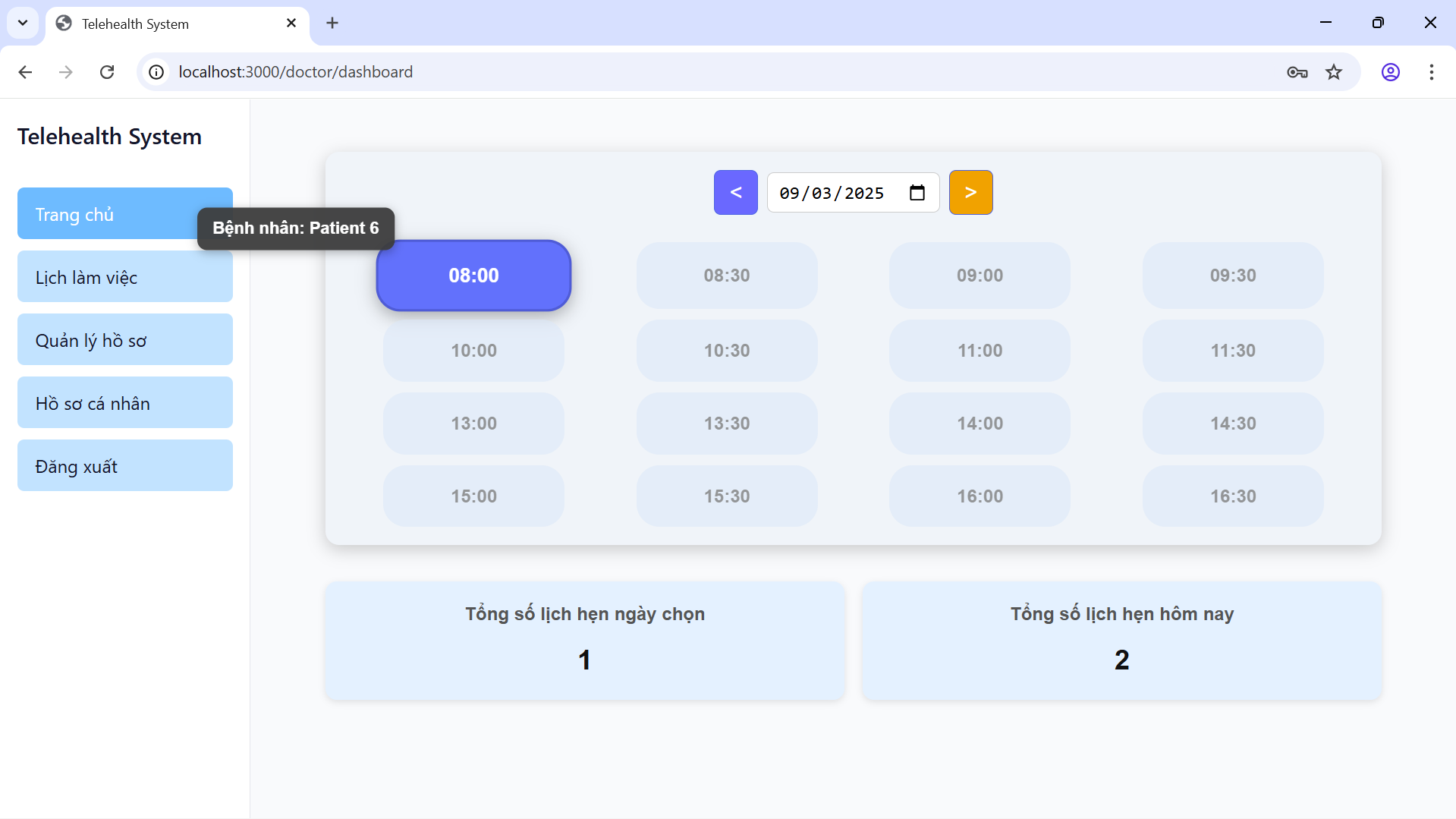
****

**E.2 Trang đặt lịch bệnh nhân**

**ApppointmentForm**

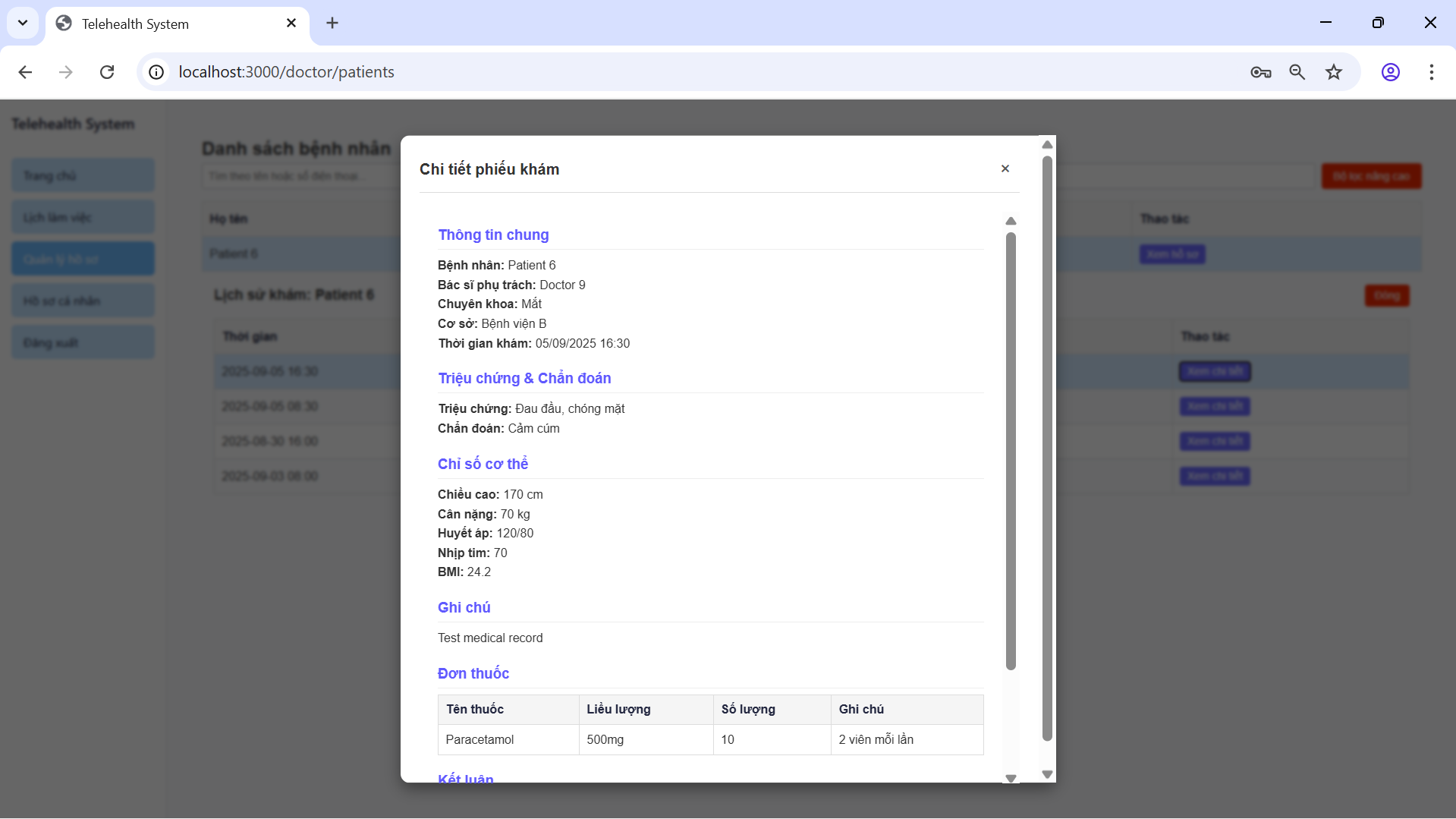
****

**E.3 Trang bác sĩ xem nhanh lịch làm việc và tương tác slots  
DoctorDashboard**

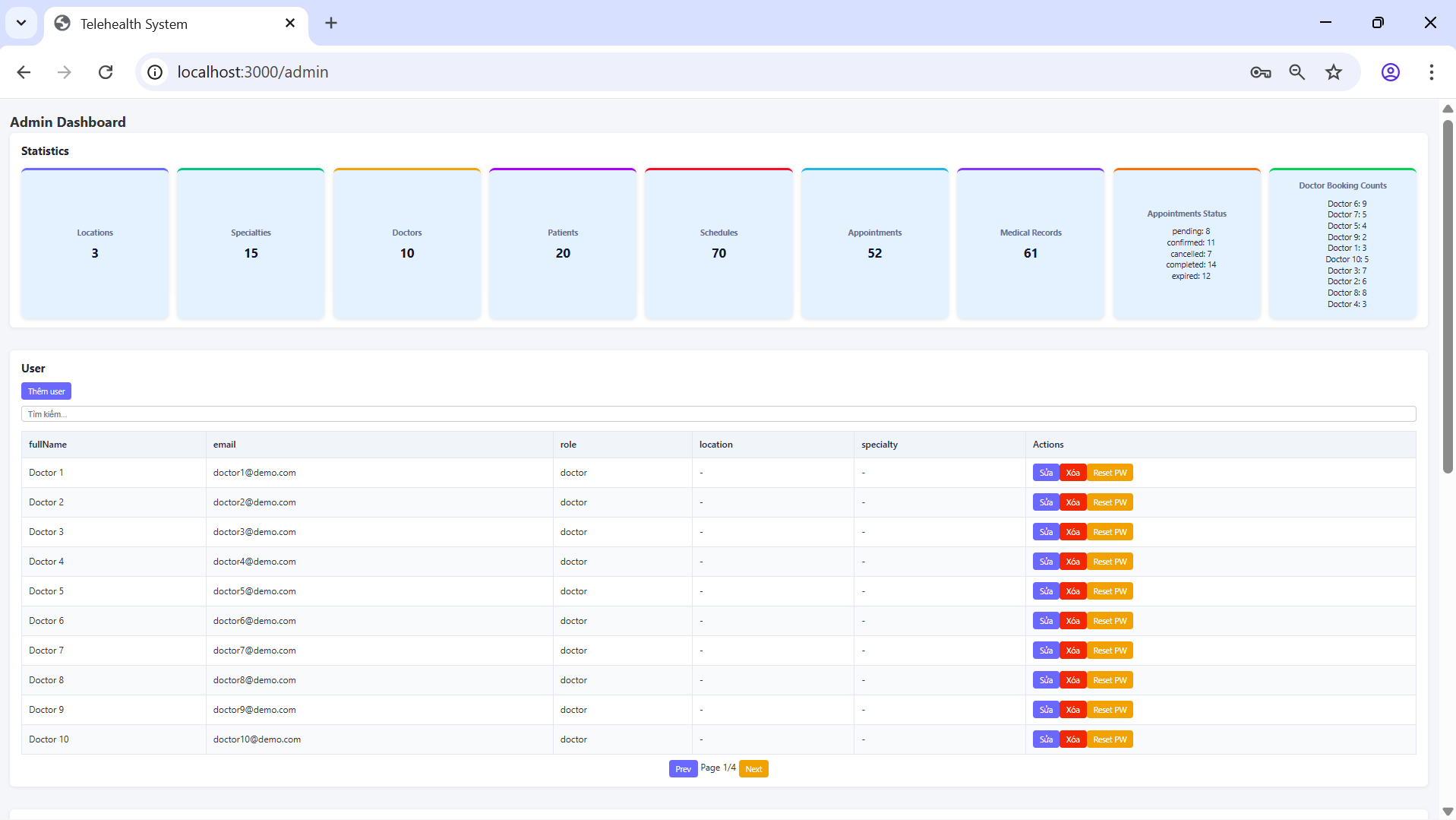
****

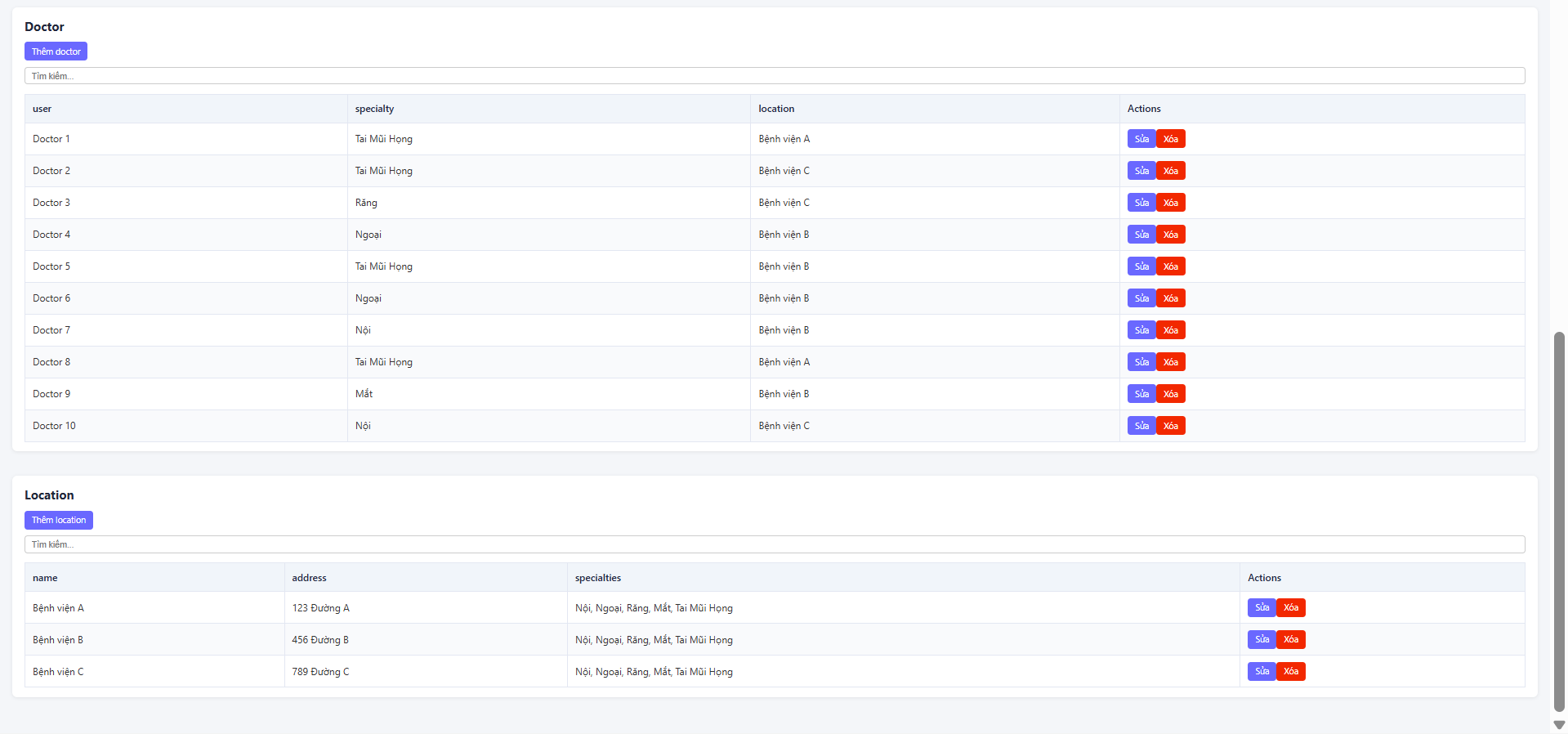
**E.4 Trang bác sĩ quản lý bệnh nhân / hồ sơ bệnh án**

**PatientList/MedicalRecordList/Detail**

****

**E.5 Trang admin quản lý người dùng / thống kê hệ thống**

****

****

Hình 5 Giao diện UI cơ bản

***// Ghi chú: Chi tiết đầy đủ trong GitHuh và Source code đính kèm***