TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO MÔN HỌC**

**CÔNG NGHỆ PHẦN MỀM**

**XÂY DỰNG WEBSITE ĐẶT LỊCH**

**KHÁM BỆNH TRỰC TUYẾN**

|  |  |
| --- | --- |
| **Giáo viên hướng dẫn:**  TS. Nguyễn Bảo Ân | **Sinh viên thực hiện:**  Nguyễn Hoài An\_110122029  Nguyễn Văn Tổng\_110122188  Kim Thạch Minh Trí\_110122191 |

***Trà Vinh, tháng năm 2025***

TRƯỜNG KỸ THUẬT VÀ CÔNG NGHỆ

KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO MÔN HỌC**

**CÔNG NGHỆ PHẦN MỀM**

**XÂY DỰNG WEBSITE ĐẶT LỊCH**

**KHÁM BỆNH TRỰC TUYẾN**

|  |  |
| --- | --- |
| **Giáo viên hướng dẫn:**  TS. Nguyễn Bảo Ân | **Sinh viên thực hiện:**  Nguyễn Hoài An\_110122029  Nguyễn Văn Tổng\_110122188  Kim Thạch Minh Trí\_110122191 |

***Trà Vinh, tháng năm 2025***

# LỜI CẢM ƠN

Trước hết, nhóm chúng em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Bảo Ân, người đã tận tình giảng dạy và hướng dẫn trong suốt quá trình học tập và thực hiện bài báo cáo này.

Chúng em cũng xin cảm ơn các thành viên trong nhóm vì sự phối hợp, tinh thần trách nhiệm và làm việc nghiêm túc để hoàn thành bài báo cáo một cách đầy đủ và đúng tiến độ. Dù còn nhiều hạn chế, nhưng chúng em đã cố gắng hết sức để bài làm phản ánh đúng năng lực và kiến thức đã tiếp thu được.

Nhóm chúng em rất mong nhận được ý kiến đóng góp từ quý thầy cô để bài báo cáo được hoàn thiện hơn.

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc204092003)

[MỤC LỤC ii](#_Toc204092004)

[DANH MỤC HÌNH ẢNH iv](#_Toc204092005)

[DANH MỤC BẢNG BIỂU v](#_Toc204092006)

[CHƯƠNG 1. TỔNG QUAN 1](#_Toc204092007)

[1.1 Tổng quan về đề tài nghiên cứu 1](#_Toc204092008)

[1.2 Lý do chọn đề tài 1](#_Toc204092009)

[1.3 Mục tiêu 1](#_Toc204092010)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 3](#_Toc204092011)

[2.1. Các chức năng chính của hệ thống (Functional Requirements) 3](#_Toc204092012)

[2.2. Các yêu cầu phi chức năng (Non-functional Requirements) 3](#_Toc204092013)

[2.3. Các mô hình phát triển phần mềm 4](#_Toc204092014)

[2.4. Tính an toàn, tính riêng tư và đảm bảo chất lượng phần mềm 4](#_Toc204092015)

[CHƯƠNG 3. THIẾT KẾ HỆ THỐNG 6](#_Toc204092016)

[3.1. Kiến trúc tổng thể 6](#_Toc204092017)

[3.2. Thiết kế cơ sở dữ liệu 6](#_Toc204092018)

[3.3. Thiết kế API 7](#_Toc204092019)

[3.4. Thiết kế giao diện người dùng 8](#_Toc204092020)

[3.5. Bản thiết kế liên kết Figma 12](#_Toc204092021)

[CHƯƠNG 4. TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG 13](#_Toc204092022)

[4.1. Danh sách các công nghệ đã sử dụng 13](#_Toc204092023)

[4.2. Quy trình CI/CD với GitHub Actions 13](#_Toc204092024)

[4.3. Cấu hình Docker và quy trình triển khai ứng dụng 14](#_Toc204092025)

[CHƯƠNG 5. QUẢN LÝ DỰ ÁN 16](#_Toc204092026)

[5.1. Lập kế hoạch và theo dõi tiến độ bằng Jira Software 16](#_Toc204092027)

[5.2. Phân công 17](#_Toc204092028)

[CHƯƠNG 6. KIỂM THỬ 19](#_Toc204092029)

[6.1. Chiến lược kiểm thử và công cụ sử dụng 19](#_Toc204092030)

[6.2. Kết quả kiểm thử 20](#_Toc204092031)

[CHƯƠNG 7. ĐÁNH GIÁ VÀ KẾT LUẬN 29](#_Toc204092032)

[7.1. Đánh giá 29](#_Toc204092033)

[7.2. Kết luận 29](#_Toc204092034)

[PHỤ LỤC 30](#_Toc204092035)

# DANH MỤC HÌNH ẢNH

[Hình 3.1 Kiến trúc tổng thể 6](#_Toc204084985)

[Hình 3.2 Giao diện trang đăng nhập 8](#_Toc204084986)

[Hình 3.3 Giao diện trang đăng ký 8](#_Toc204084987)

[Hình 3.4 Giao diện trang dashboard bệnh nhân 9](#_Toc204084988)

[Hình 3.5 Giao diện trang các chuyên khoa 9](#_Toc204084989)

[Hình 3.6 Giao diện chi tiết trang chuyên khoa 9](#_Toc204084990)

[Hình 3.7 Giao diện đặt lịch khám của bệnh nhân 10](#_Toc204084991)

[Hình 3.8 Giao diện lịch khám bệnh của bác sĩ 10](#_Toc204084992)

[Hình 3.9 Giao diện trang Dashboard của Admin 10](#_Toc204084993)

[Hình 3.10 Giao diện trang quản lý user 11](#_Toc204084994)

[Hình 3.11 Giao diện trang quản lý bác sĩ 11](#_Toc204084995)

[Hình 3.12 Giao diện trang quản lý chuyên khoa 11](#_Toc204084996)

[Hình 3.13 Giao diện trang quản lý các lịch khám 12](#_Toc204084997)

[Hình 5.1 Quản lý hồ sơ và xem danh sách bác sĩ theo chuyên khoa 16](#_Toc204084998)

[Hình 5.2 Đăng nhập/đăng ký, đặt lịch, xem thông tin 16](#_Toc204084999)

[Hình 5.3 Thay đổi/hủy, xác nhận/từ chối, câp nhật trạng thái lịch hẹn 17](#_Toc204085000)

[Hình 5.4 Biểu đồ Velocity của các Sprint, thể hiện năng suất hoàn thành công việc 17](#_Toc204085001)

[Hình 6.1 Cấu hình Header trong Postman 19](#_Toc204085002)

[Hình 6.2 Gửi yêu cầu đăng nhập với JSON Body 20](#_Toc204085003)

[Hình 6.3 Phản hồi từ API đăng nhập – trả về token và thông tin người dùng 20](#_Toc204085004)

[Hình 6.4 Gửi yêu cầu đăng ký tài khoản 21](#_Toc204085005)

[Hình 6.5 Kết quả trả về - đăng ký tài khoản 21](#_Toc204085006)

[Hình 6.6 Gửi yêu cầu đặt lịch với token xác thực 21](#_Toc204085007)

[Hình 6.7 Gửi dữ liệu đặt lịch dưới dạng JSON 22](#_Toc204085008)

[Hình 6.8 Kết quả trả về khi đặt lịch thành công từ Postman 22](#_Toc204085009)

[Hình 6.9 Kết quả trả về danh sách bác sĩ 23](#_Toc204085010)

[Hình 6.10 Kết quả trả về danh sách các chuyên khoa 23](#_Toc204085011)

# DANH MỤC BẢNG BIỂU

[Bảng 3.1 Danh sách các bảng trong cơ sở dữ liệu và vai trò tương ứng 7](#_Toc204085012)

[Bảng 4.1 Danh sách các công nghệ/công cụ đã sử dụng trong đề tài 13](#_Toc204085013)

[Bảng 4.2 CI/CD với GitHub Actions 13](#_Toc204085014)

[Bảng 5.1 Bảng phân công công việc 18](#_Toc204085015)

# TỔNG QUAN

## Tổng quan về đề tài nghiên cứu

Tên đề tài: Xây dựng Website đặt lịch khám bệnh trực tuyến.

Chủ đề nghiên cứu: Ứng dụng công nghệ thông tin vào ngành y tế – chăm sóc sức khỏe.

## Lý do chọn đề tài

Tình hình thực tế: Tại nhiều bệnh viện, việc đăng ký khám bệnh còn thủ công, gây nên tình trạng quá tải, người bệnh phải chờ đợi rất lâu, thậm chí không được khám nếu số lượng bệnh nhân quá đông.

**Mục đích thực hiện đề tài:**

Tăng cường ứng dụng công nghệ thông tin trong lĩnh vực y tế: Việc ứng dụng công nghệ để giảm tải hành chính, tăng cường phục vụ người dân là một xu thế tất yếu trong chuyển đổi số y tế.

Tính thực tiễn cao: Đề tài mang tính ứng dụng thực tế cao, có thể triển khai cho phòng khám tư nhân, bệnh viện công lập hoặc các hệ thống chăm sóc sức khỏe cộng đồng.

**Cơ hội học tập kỹ năng toàn diện:**

Kết hợp nhiều công nghệ như React, NodeJS, MySQL, Docker, CI/CD.

Rèn luyện kỹ năng làm việc nhóm, sử dụng công cụ quản lý dự án Jira, viết tài liệu chuyên nghiệp.

Trải nghiệm quy trình phát triển phần mềm từ phân tích yêu cầu đến thiết kế, triển khai, kiểm thử và đánh giá.

Ngoài ra, xu hướng chuyển đổi số y tế đòi hỏi việc áp dụng các công nghệ hiện đại như phần mềm phân tán, microservices và đám mây để đảm bảo hệ thống có thể mở rộng quy mô, xử lý lượng lớn người dùng đồng thời, và giảm tải cho máy chủ cục bộ. Đặc biệt, microservices cho phép tách biệt các chức năng (như xác thực, đặt lịch) thành các dịch vụ độc lập, tăng tính linh hoạt và dễ bảo trì, trong khi triển khai trên đám mây (như AWS hoặc Google Cloud) giúp tối ưu chi phí và tăng tính sẵn sàng, phù hợp với nhu cầu thực tế của các cơ sở y tế.

## Mục tiêu

Mục tiêu thực hiện đề tài "Xây dựng Website đặt lịch khám bệnh trực tuyến" nhằm mục đích số hóa quá trình đặt lịch hẹn khám bệnh giữa bệnh nhân và cơ sở y tế.

Ứng dụng cung cấp nền tảng trung gian giúp:

Người bệnh dễ dàng tra cứu thông tin bác sĩ, chuyên khoa và các khung giờ trống để đặt lịch khám bệnh trực tuyến nhanh chóng.

Bác sĩ có thể xem, xác nhận hoặc từ chối các lịch hẹn, tối ưu hóa thời gian làm việc, tránh quá tải hoặc lãng phí thời gian trống.

Quản trị viên có thể quản lý hệ thống, quản lý người dùng, cập nhật thông tin bác sĩ, chuyên khoa và kiểm soát hoạt động đặt lịch.

Xây dựng một nền tảng đơn giản, dễ dùng, dễ triển khai, có thể mở rộng quy mô và tích hợp thêm các tính năng như thông báo, thanh toán online, hoặc tư vấn khám bệnh từ xa.

# CƠ SỞ LÝ THUYẾT

## Các chức năng chính của hệ thống (Functional Requirements)

Đề tài "Xây dựng Website đặt lịch khám bệnh trực tuyến" được xây dựng xoay quanh 3 vai trò chính: Người dùng (bệnh nhân), Bác sĩ, và Quản trị viên (Admin). Các chức năng chính bao gồm:

**Đối với Người dùng:**

* Đăng ký, đăng nhập và đăng xuất: Cho phép người dùng tạo tài khoản, đăng nhập để sử dụng dịch vụ và đăng xuất để đảm bảo bảo mật.
* Cập nhật thông tin cá nhân: Người dùng có thể chỉnh sửa thông tin như họ tên, số điện thoại, ngày sinh,...
* Tìm kiếm và xem danh sách bác sĩ theo chuyên khoa: Người dùng có thể lọc bác sĩ dựa trên chuyên khoa để lựa chọn phù hợp.
* Đặt lịch khám: Chọn bác sĩ, chuyên khoa, ngày giờ khám bệnh dựa trên lịch trống của bác sĩ.

Quản lý lịch hẹn: Cho phép người dùng hủy hoặc thay đổi lịch hẹn khi cần thiết.

**Đối với Bác sĩ:**

* Xem danh sách lịch hẹn của mình: Giúp bác sĩ biết được lịch làm việc để sắp xếp thời gian hợp lý.
* Xác nhận hoặc từ chối lịch hẹn: Giúp kiểm soát và tối ưu hóa thời gian làm việc.
* Cập nhật hồ sơ cá nhân: Chỉnh sửa thông tin chuyên môn, thời gian làm việc,...

**Đối với Quản trị viên:**

* Quản lý tài khoản người dùng và bác sĩ: Có thể khóa/mở tài khoản hoặc chỉnh sửa thông tin.
* Quản lý chuyên khoa, bệnh viện: Cập nhật thông tin hệ thống.
* Theo dõi thống kê lịch hẹn, hoạt động hệ thống.

## Các yêu cầu phi chức năng (Non-functional Requirements)

Giao diện thân thiện, dễ sử dụng: Thiết kế tối giản, rõ ràng cho cả người dùng phổ thông và bác sĩ.

Ứng dụng chạy được trên cả máy tính và thiết bị di động.

Hiệu năng tốt: Thời gian tải trang và xử lý đặt lịch phải nhanh, đặc biệt trong khung giờ cao điểm.

Tính bảo mật:

* Mật khẩu được mã hóa.
* Chỉ người dùng đã đăng nhập mới được truy cập các chức năng đặt lịch.
* Phân quyền rõ ràng theo vai trò.

Tính mở rộng: Hệ thống có khả năng mở rộng thêm các chức năng như thanh toán online, nhắc lịch qua email,...

Tính khả dụng cao: Luôn sẵn sàng phục vụ người dùng, ít bị gián đoạn khi triển khai thực tế.

## Các mô hình phát triển phần mềm

Để xây dựng hệ thống "Xây dựng Website đặt lịch khám bệnh trực tuyến", việc lựa chọn mô hình phát triển phần mềm phù hợp là rất quan trọng. Dưới đây là các mô hình phổ biến và cách áp dụng vào dự án:

* Mô hình Thác nước (Waterfall Model): Quy trình tuyến tính với các giai đoạn tuần tự (phân tích, thiết kế, triển khai, kiểm thử, bảo trì). Phù hợp với dự án có yêu cầu cố định nhưng không linh hoạt khi thay đổi.
* Mô hình Agile: Tập trung vào phát triển lặp qua các sprint, ưu tiên phản hồi nhanh và sự hợp tác với người dùng. Scrum, một framework của Agile, được sử dụng trong dự án này.
* Mô hình Xoắn ốc (Spiral Model): Kết hợp phát triển lặp và quản lý rủi ro, phù hợp với dự án phức tạp nhưng yêu cầu nguồn lực cao.

Lý do chọn Agile Scrum: Dự án yêu cầu linh hoạt để điều chỉnh giao diện và chức năng dựa trên phản hồi từ người dùng và giảng viên. Scrum giúp chia nhỏ công việc thành các sprint, đảm bảo tiến độ và chất lượng.

## Tính an toàn, tính riêng tư và đảm bảo chất lượng phần mềm

Hệ thống đòi hỏi các tiêu chuẩn cao về an toàn, riêng tư và chất lượng. Các khái niệm chính bao gồm:

* Tính an toàn (Security): Bảo vệ hệ thống khỏi truy cập trái phép qua xác thực (JWT), phân quyền, và mã hóa dữ liệu.
* Tính riêng tư (Privacy): Đảm bảo thông tin cá nhân (email, tên) chỉ được sử dụng cho mục đích đặt lịch, cần chính sách bảo mật rõ ràng.
* Đảm bảo chất lượng (Quality Assurance): Đảm bảo phần mềm đúng chức năng, hiệu năng cao và dễ bảo trì.

Ứng dụng trong dự án: Hệ thống sử dụng JWT và mã hóa mật khẩu nhưng cần bổ sung chính sách riêng tư và kiểm thử tải để nâng cao chất lượng.

# THIẾT KẾ HỆ THỐNG

## Kiến trúc tổng thể

Hệ thống được xây dựng theo mô hình Client-Server.

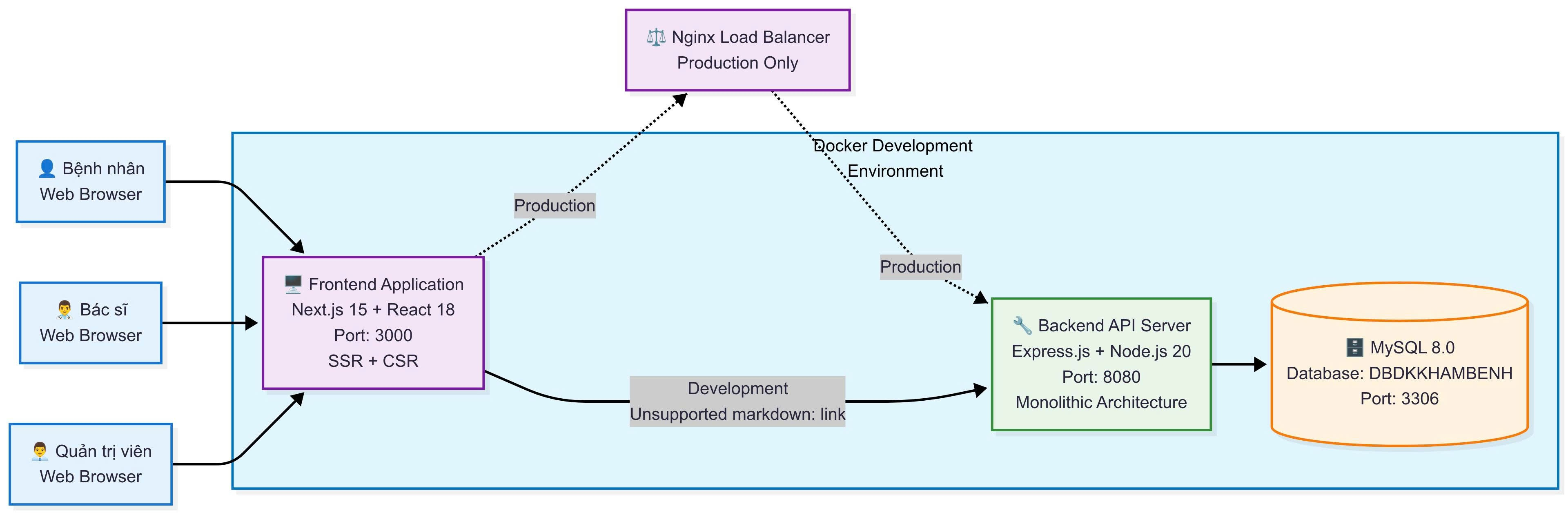
Client là ứng dụng giao diện người dùng, phát triển bằng Next.js – một framework hiện đại dựa trên React.

Server là hệ thống backend được viết bằng Node.js kết hợp Express.js, xử lý các logic nghiệp vụ và kết nối cơ sở dữ liệu.

Dữ liệu được lưu trữ trong hệ quản trị MySQL, tương tác thông qua Sequelize.

Giao tiếp giữa frontend và backend được thực hiện thông qua REST API, dưới định dạng JSON.

Hệ thống được triển khai bằng Docker, mỗi thành phần chạy trong một container riêng biệt. Docker giúp đơn giản hoá quá trình cài đặt, đảm bảo môi trường đồng nhất từ phát triển đến triển khai thực tế.



Hình 3.1 Kiến trúc tổng thể

Hệ thống được xây dựng theo mô hình Client-Server, có thể tích hợp microservices trong tương lai. Microservices cho phép chia nhỏ hệ thống thành các dịch vụ độc lập (xác thực, quản lý lịch hẹn, quản lý người dùng), mỗi dịch vụ chạy trong container Docker riêng biệt, tăng khả năng mở rộng và bảo trì. Hiện tại, hệ thống sử dụng kiến trúc đơn khối (monolith) với Node.js/Express, nhưng có thể chuyển đổi sang microservices bằng cách tách các endpoint API thành các dịch vụ riêng.

## Thiết kế cơ sở dữ liệu

Hệ thống sử dụng cơ sở dữ liệu quan hệ với các bảng sau:

Bảng 3.1 Danh sách các bảng trong cơ sở dữ liệu và vai trò tương ứng

| **STT** | **Tên bảng** | **Vai trò** |
| --- | --- | --- |
| 1 | User | Lưu thông tin người dùng (bệnh nhân, bác sĩ, quản trị) |
| 2 | DoctorDetail | Thông tin chi tiết của bác sĩ: chuyên khoa, mô tả, giá khám |
| 3 | Schedule | Lịch trình của bác sĩ: ngày, giờ, số lượng bệnh nhân tối đa |
| 4 | Booking | Các lịch hẹn mà bệnh nhân đã đặt |
| 5 | Specialty | Danh sách chuyên khoa y tế |
| 6 | Allcode | Bảng mã hóa dùng chung cho hệ thống: giới tính, vai trò, trạng thái, khung giờ,... |

Quan hệ giữa các bảng:

Một người dùng (bác sĩ) có thể có một bản ghi DoctorDetail.

Một bác sĩ có thể có nhiều lịch trình (Schedule), và mỗi lịch có thể có nhiều lịch đặt khám (Booking).

Mỗi Booking liên kết với một người dùng (bệnh nhân) và một lịch trình (Schedule).

## Thiết kế API

Backend cung cấp các API REST để frontend tương tác. Dưới đây là các nhóm endpoint chính:

1. **Xác thực:**

POST /api/auth/login: Đăng nhập, trả về token JWT.

GET /api/auth/profile: Truy xuất thông tin người dùng từ token.

1. **Bác sĩ và chuyên khoa:**

GET /api/specialties: Lấy danh sách chuyên khoa.

GET /api/doctors: Lấy danh sách bác sĩ.

GET /api/doctors/:id: Thông tin chi tiết bác sĩ.

1. **Lịch trình và đặt khám:**

GET /api/schedule/:doctorId: Lấy lịch trình bác sĩ.

POST /api/schedule: Tạo mới lịch trình.

POST /api/booking: Bệnh nhân đặt lịch khám.

GET /api/bookings/:userId: Lịch sử đặt khám của người dùng.

1. **Quản lý hệ thống (Admin):**

GET /api/users: Lấy danh sách người dùng.

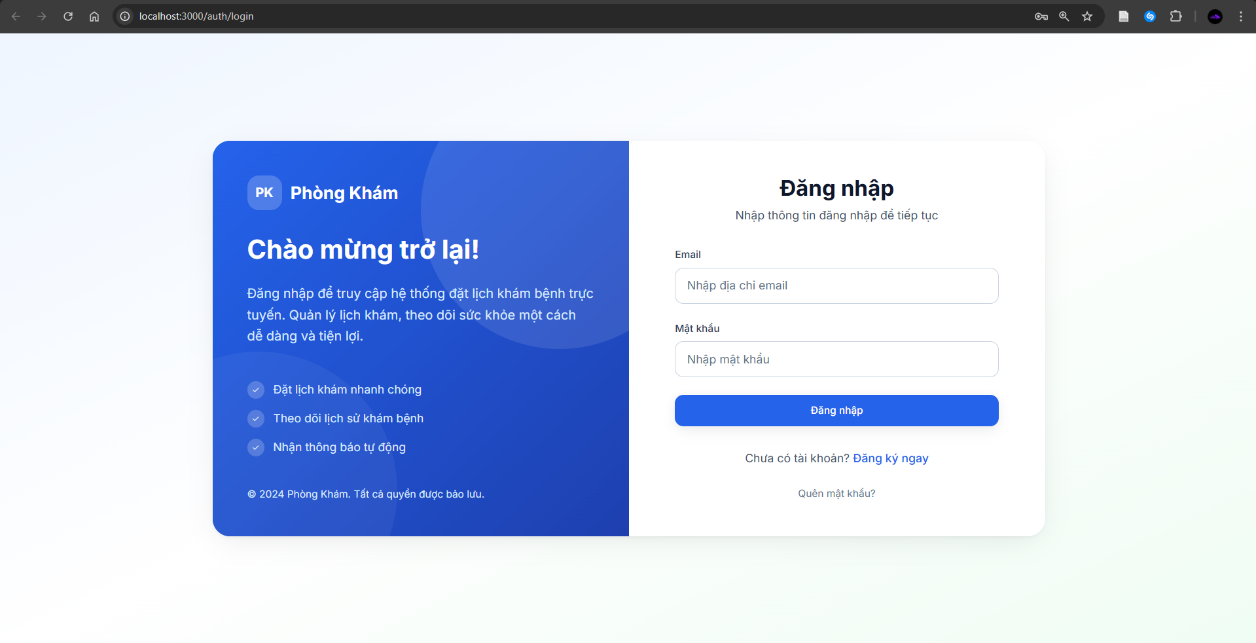
POST /api/users: Thêm người dùng.

DELETE /api/users/:id: Xoá người dùng.

Tất cả các endpoint bảo mật đều sử dụng JWT và được kiểm tra bởi middleware verifyToken.

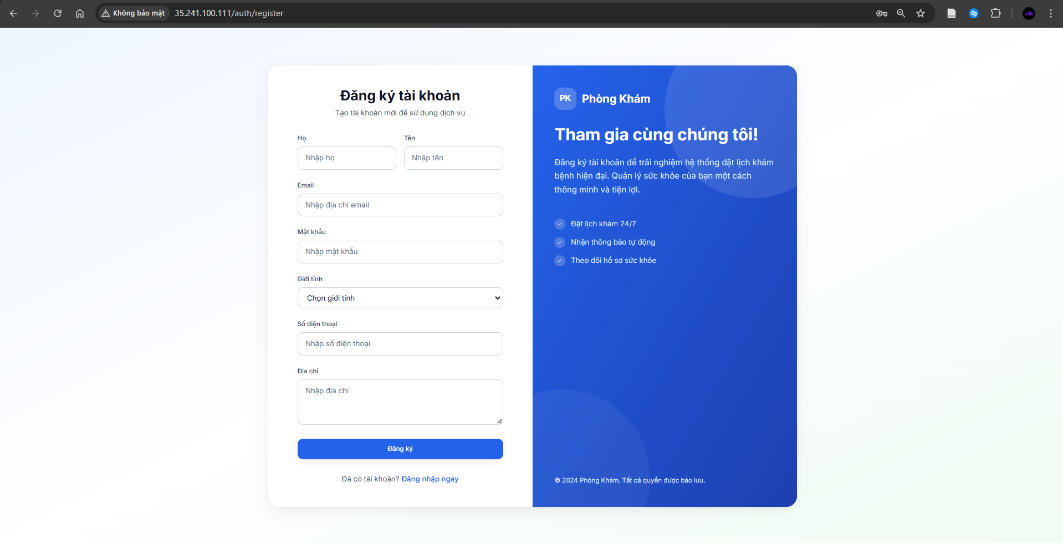
## Thiết kế giao diện người dùng

### Giao diện trang đăng nhập



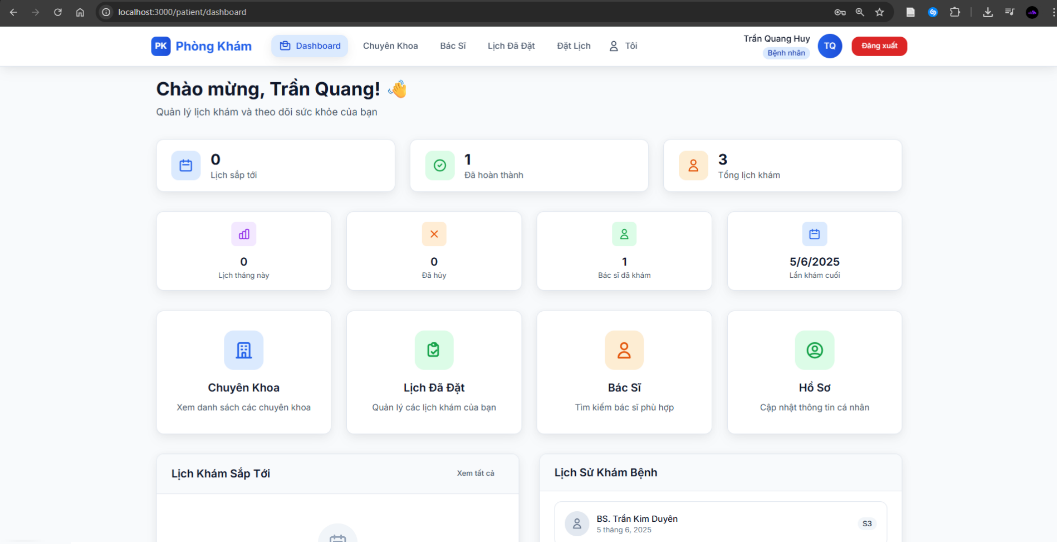
Hình 3.2 Giao diện trang đăng nhập

### Giao diện trang đăng ký



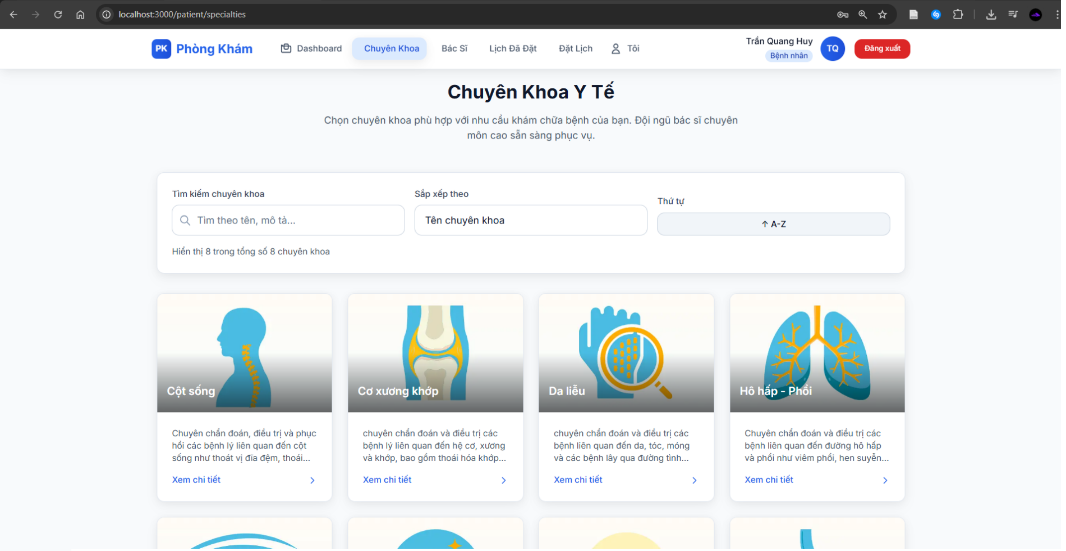
Hình 3.3 Giao diện trang đăng ký

### Giao diện trang dashboard bệnh nhân



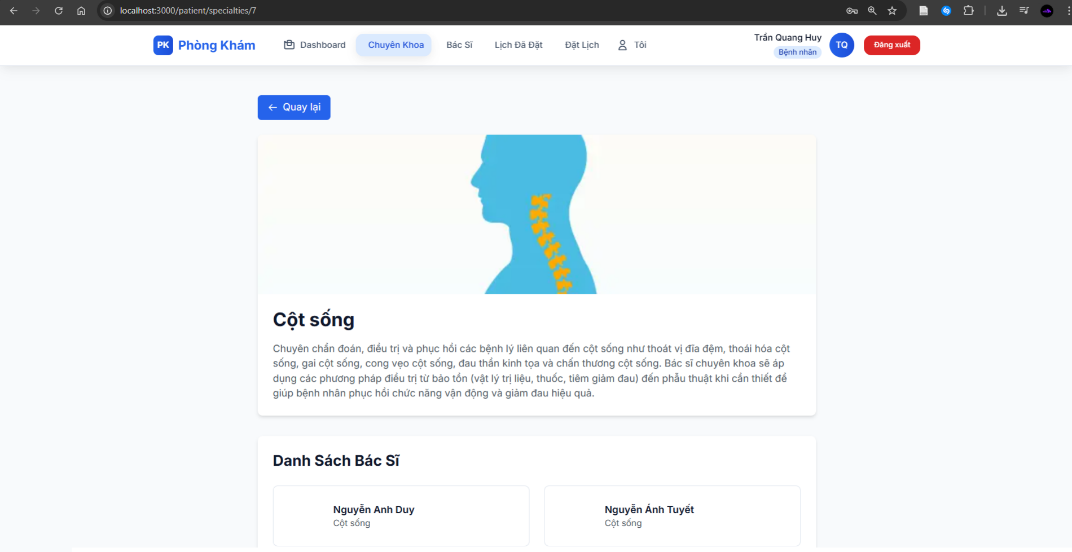
Hình 3.4 Giao diện trang dashboard bệnh nhân

### Giao diện trang các chuyên khoa



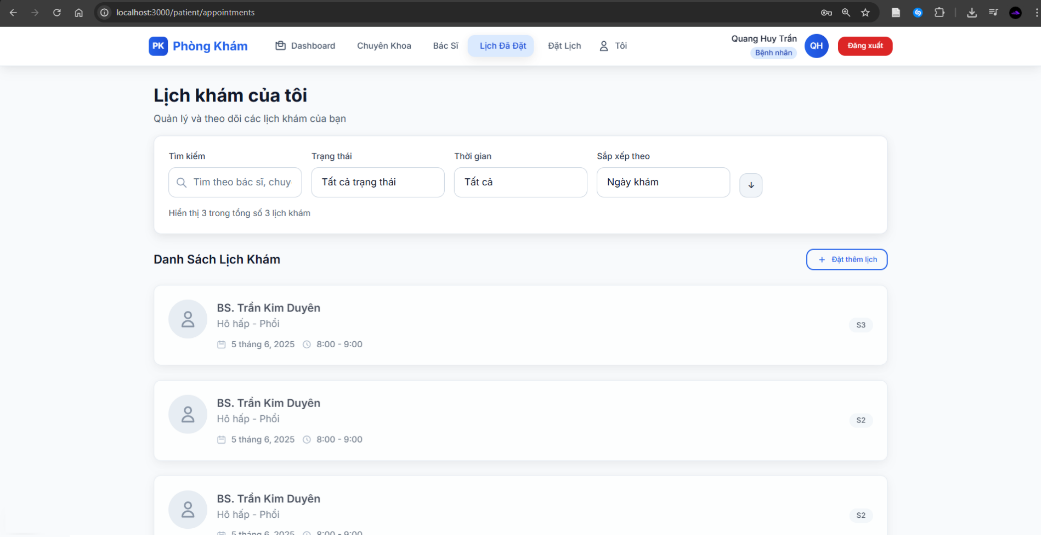
Hình 3.5 Giao diện trang các chuyên khoa

### Giao diện chi tiết trang chuyên khoa



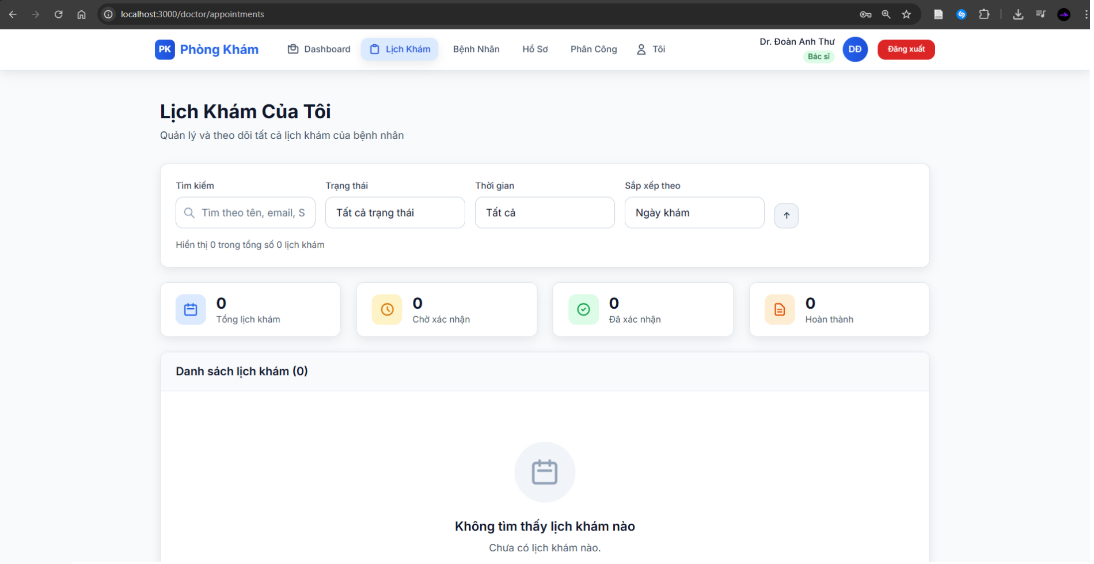
Hình 3.6 Giao diện chi tiết trang chuyên khoa

### Giao diện đặt lịch khám của bệnh nhân



Hình 3.7 Giao diện đặt lịch khám của bệnh nhân

### Giao diện lịch khám bệnh của bác sĩ



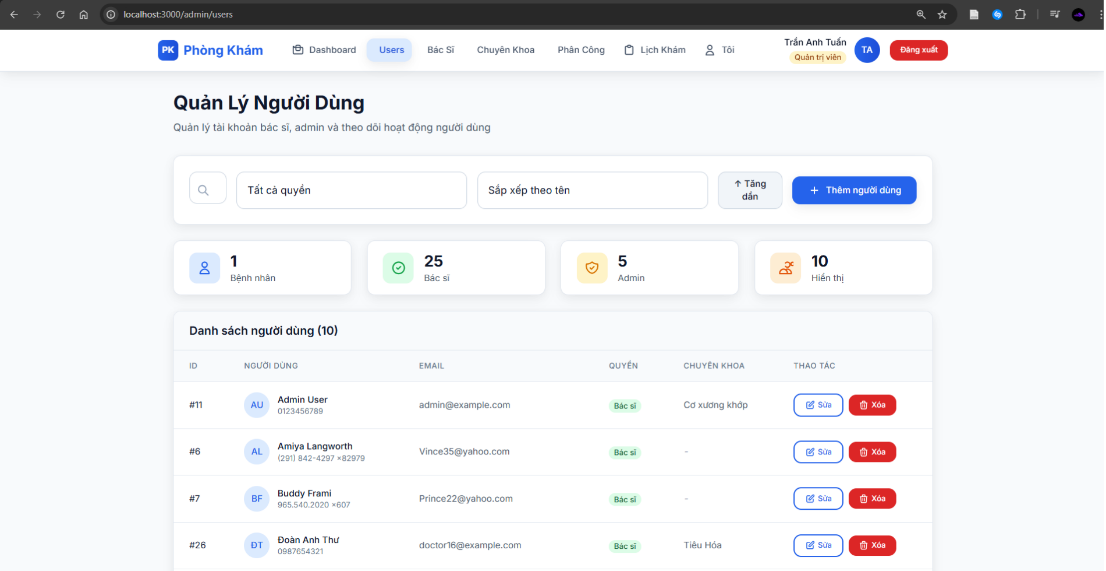
Hình 3.8 Giao diện lịch khám bệnh của bác sĩ

### Giao diện trang Dashboard của Admin



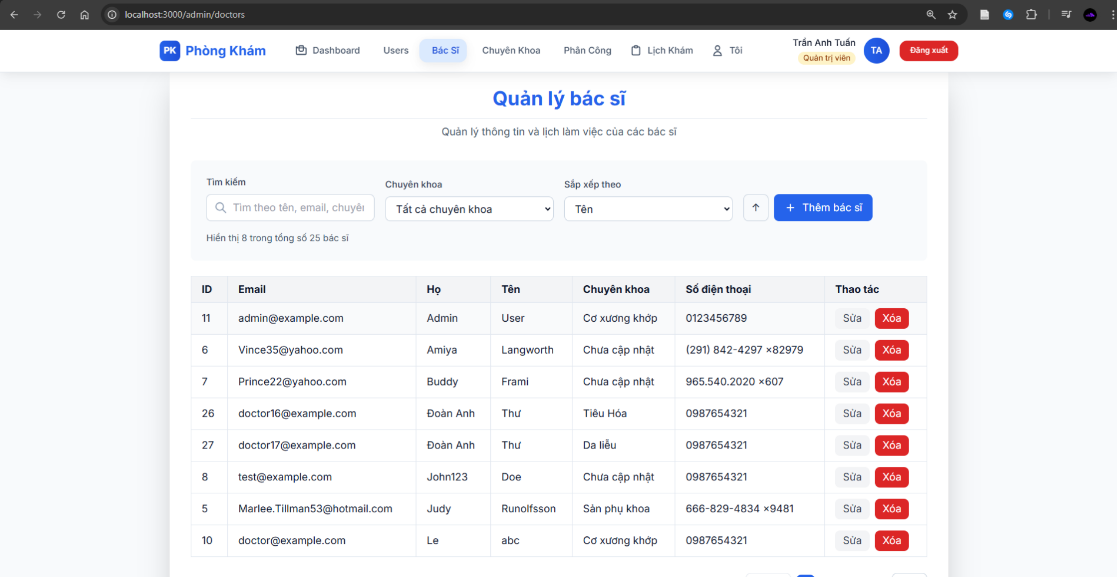
Hình 3.9 Giao diện trang Dashboard của Admin

### Giao diện trang quản lý user của Admin



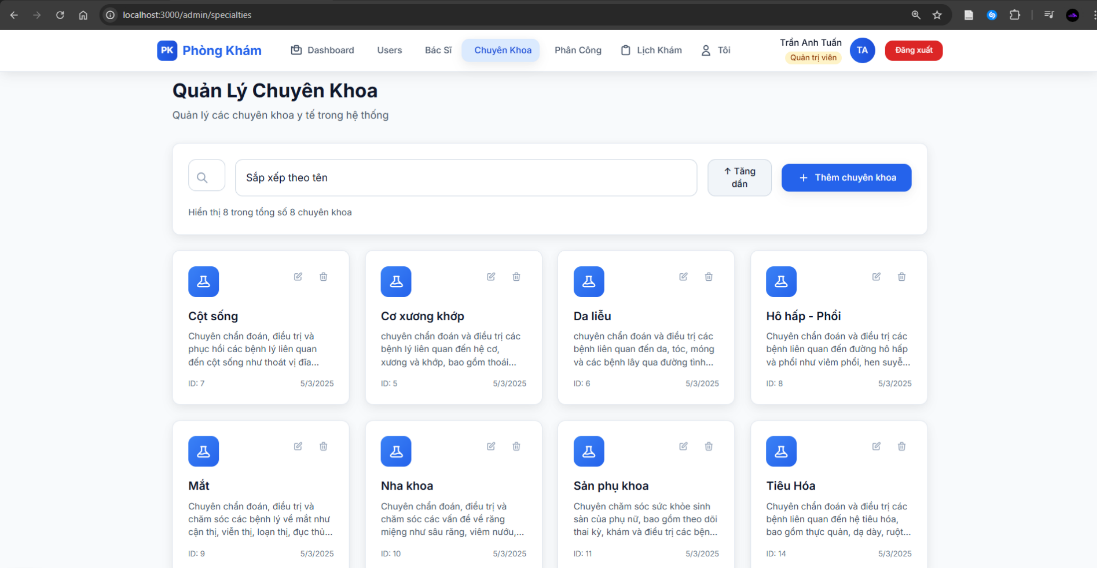
Hình 3.10 Giao diện trang quản lý user

### Giao diện trang quản lý bác sĩ của Admin



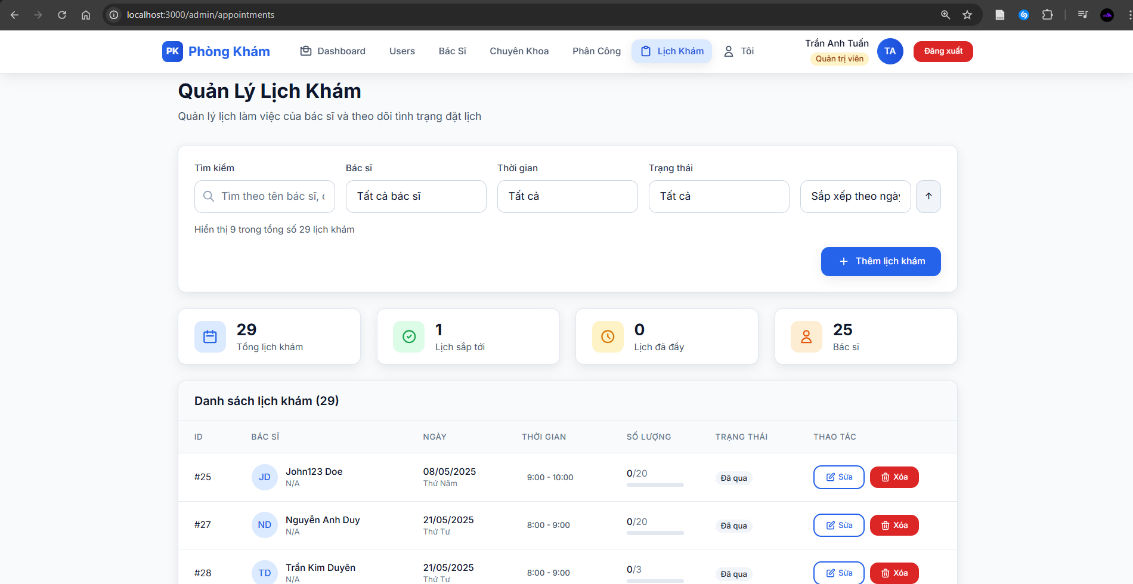
Hình 3.11 Giao diện trang quản lý bác sĩ

### Giao diện trang quản lý chuyên khoa



Hình 3.12 Giao diện trang quản lý chuyên khoa

### Giao diện trang quản lý các lịch khám



Hình 3.13 Giao diện trang quản lý các lịch khám

## Bản thiết kế liên kết Figma

Để đảm bảo tính trực quan và dễ sử dụng của hệ thống, nhóm chúng tôi đã thiết kế giao diện người dùng (UI) thông qua công cụ Figma. Bản thiết kế thể hiện đầy đủ các màn hình chính của hệ thống như:

* Trang đăng nhập, đăng ký
* Trang chuyên khoa
* Trang đặt lịch khám
* Trang thông tin bác sĩ
* Trang quản lý của Admin

Người xem có thể truy cập và duyệt trực tiếp bản thiết kế tại liên kết sau:

<https://www.figma.com/design/tNJFzRt9esyP22epLfQ6zO/CNPM?node-id=0-1&t=wcTlt7DxQMbNkqHk-1>

# TRIỂN KHAI VÀ CÔNG NGHỆ SỬ DỤNG

## Danh sách các công nghệ đã sử dụng

Dưới đây là danh sách các công nghệ/công cụ đã sử dụng trong đề tài này:

Bảng 4.1 Danh sách các công nghệ/công cụ đã sử dụng trong đề tài

|  |  |  |
| --- | --- | --- |
| **STT** | **THÀNH PHẦN** | **CÔNG NGHỆ / CÔNG CỤ** |
| 1 | Thiết kế giao diện | - Figma |
| 2 | Giao diện người dùng (Frontend) | - ReactJS  - Tailwind CSS |
| 3 | Backend (API) | - Node.js  - Express |
| 4 | Cơ sở dữ liệu | MySQL |
| 5 | Quản lý mã nguồn | - GitHub  - Git |
| 6 | Quản lý dự án | - Jira (Scrum + Backlog + Sprint) |
| 7 | Tài liệu API | - Swagger |
| 8 | Kiểm thử API | - Postman |
| 9 | Triển khai | - Docker  - Docker Compose |
| 10 | Tự động hóa | - GitHub Actions (CI/CD) |
| 11 | Ngôn ngữ lập trình (sử dụng cho cả frontend và backend) | - JavaScript |

## Quy trình CI/CD với GitHub Actions

Áp dụng quy trình CI/CD thông qua GitHub Actions – giúp tự động hóa kiểm thử, kiểm tra chất lượng mã nguồn và triển khai khi có cập nhật.

1. CI/CD Pipeline (ci-cd.yml)

Kích hoạt khi: Có push lên nhánh main/develop, hoặc Pull Request

* Kiểm thử Frontend & Backend
* Quét bảo mật
* Kiểm tra phụ thuộc
* Build & Push Docker
* Triển khai tự động

1. Kiểm tra Pull Request (pr-check.yml)

Kích hoạt khi: Có Pull Request vào nhánh main hoặc develop

* Phân tích chất lượng mã
* Kiểm thử toàn diện
* Xác thực bảo mật
* Kiểm thử Docker build
* Tạo báo cáo tự động cho PR

1. Quản lý phát hành (release.yml)

Kích hoạt khi: Có tag phiên bản hoặc chạy thủ công

* Kiểm tra bản phát hành
* Build bản production
* Tạo changelog tự động
* Tạo bản phát hành GitHub
* Triển khai production

## Cấu hình Docker và quy trình triển khai ứng dụng

Docker là một công cụ quan trọng giúp đóng gói ứng dụng thành các "container" (các môi trường độc lập), đảm bảo hệ thống chạy ổn định trên mọi nền tảng.

#### Cấu hình file docker-compose.yml

Hệ thống được cấu hình với ba container:

* Frontend container: Chạy ứng dụng React trên cổng 3000 (trình duyệt người dùng truy cập).
* Backend container: Chạy Express trên cổng 8080 (xử lý logic và API).
* Database container: Sử dụng MySQL hoặc Postgres, tùy theo cấu hình.

File này định nghĩa cách các container giao tiếp và khởi động cùng nhau.

#### Quy trình triển khai

Chuẩn bị môi trường: Cài đặt Docker và Docker Compose trên máy tính.

Tạo file .env: Định nghĩa các biến môi trường (ví dụ: thông tin kết nối MySQL, khóa JWT) để bảo mật và linh hoạt.

Chạy ứng dụng: Sử dụng lệnh sau để xây dựng và khởi động các container.

docker-compose up --build

Quy trình hoạt động:

* Docker build frontend và backend thành container riêng biệt.
* Khởi chạy MySQL với cấu hình định sẵn.
* Tạo môi trường thống nhất, dễ bảo trì và triển khai trên các nền tảng khác.

#### Triển khai trên đám mây

Triển khai cục bộ: Chạy câu lệnh sau tại thư mục gốc trên máy cá nhân:

docker-compose up --build

Triển khai đám mây: Có thể sử dụng AWS Elastic Beanstalk để host container, với frontend trên cổng 3000, backend trên cổng 8080, và kết nối với MySQL trên dịch vụ đám mây như AWS RDS. Ví dụ, cấu hình Elastic Beanstalk yêu cầu file Dockerrun.aws.json để định nghĩa container.

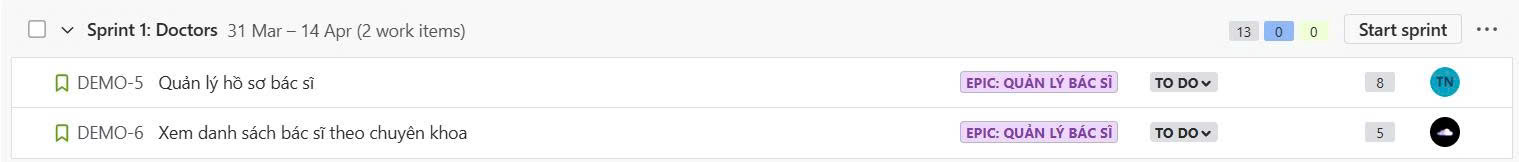
Trong tương lai: Tích hợp GitHub Actions với AWS CodePipeline để tự động triển khai mỗi khi có cập nhật mã nguồn, đảm bảo hệ thống luôn cập nhật.

# QUẢN LÝ DỰ ÁN

## Lập kế hoạch và theo dõi tiến độ bằng Jira Software

Để quản lý tiến độ và phân công công việc hiệu quả cho nhóm, chúng tôi đã sử dụng Jira Software – một công cụ quản lý dự án phổ biến theo phương pháp Agile Scrum. Tại đây, toàn bộ backlog và nhiệm vụ được phân chia thành các Epic rõ ràng, mỗi Epic bao gồm các task cụ thể ứng với nhu cầu thực tế của người dùng (user stories).

1. **Doctor:**

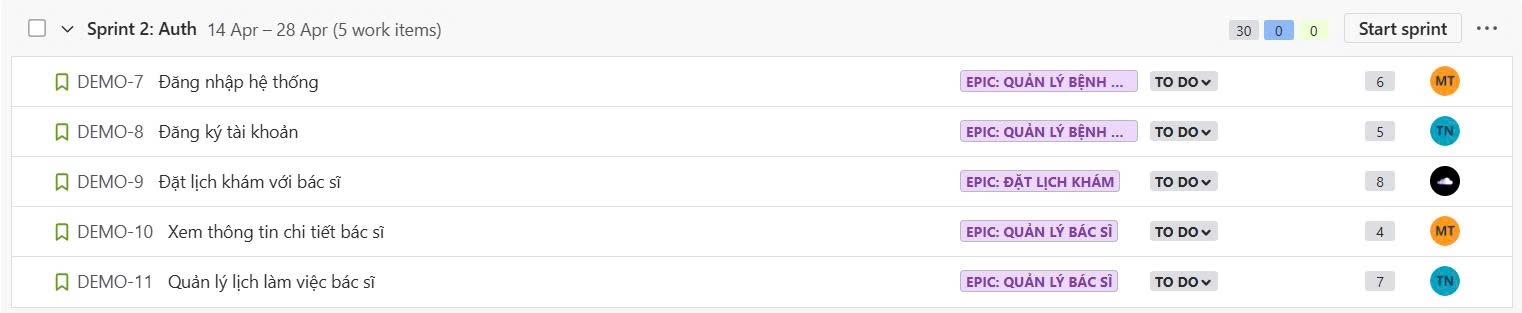


Hình 5.1 Quản lý hồ sơ và xem danh sách bác sĩ theo chuyên khoa

Mô tả Sprint: Doctors

* Là một bác sĩ, tôi muốn quản lý hồ sơ cá nhân và lịch làm việc, nhờ đó tôi có thể hiển thị thông tin chính xác cho bệnh nhân.
* Là một người dùng, tôi muốn xem danh sách bác sĩ theo chuyên khoa, nhờ đó tôi có thể chọn bác sĩ phù hợp.

1. **Auth:**

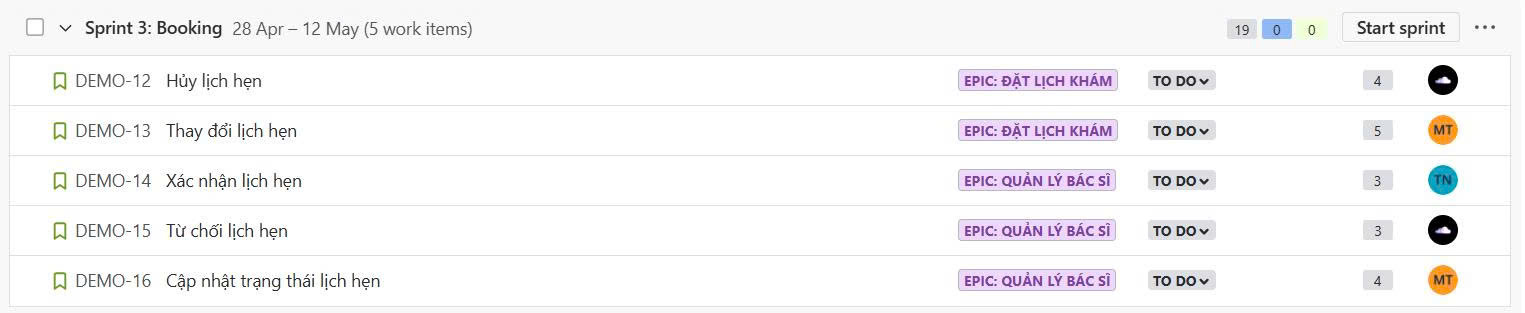


Hình 5.2 Đăng nhập/đăng ký, đặt lịch, xem thông tin

Mô tả Sprint: Auth

* Là một người dùng, tôi muốn đăng nhập/đăng xuất, nhờ đó tôi có thể bảo vệ thông tin cá nhân của mình.
* Là một người dùng mới, tôi muốn đăng ký tài khoản, nhờ đó tôi có thể sử dụng hệ thống để đặt lịch khám.
* Là một người dùng, tôi muốn chọn ngày giờ và đặt lịch khám với bác sĩ, nhờ đó tôi có thể đảm bảo có lịch hẹn phù hợp.
* Là một người dùng, tôi muốn xem thông tin chi tiết bác sĩ để đưa ra quyết định đặt lịch.
* Là một bác sĩ, tôi muốn thiết lập lịch làm việc để bệnh nhân có thể đặt lịch phù hợp.

1. **Quản lý lịch hẹn (Booking):**



Hình 5.3 Thay đổi/hủy, xác nhận/từ chối, câp nhật trạng thái lịch hẹn

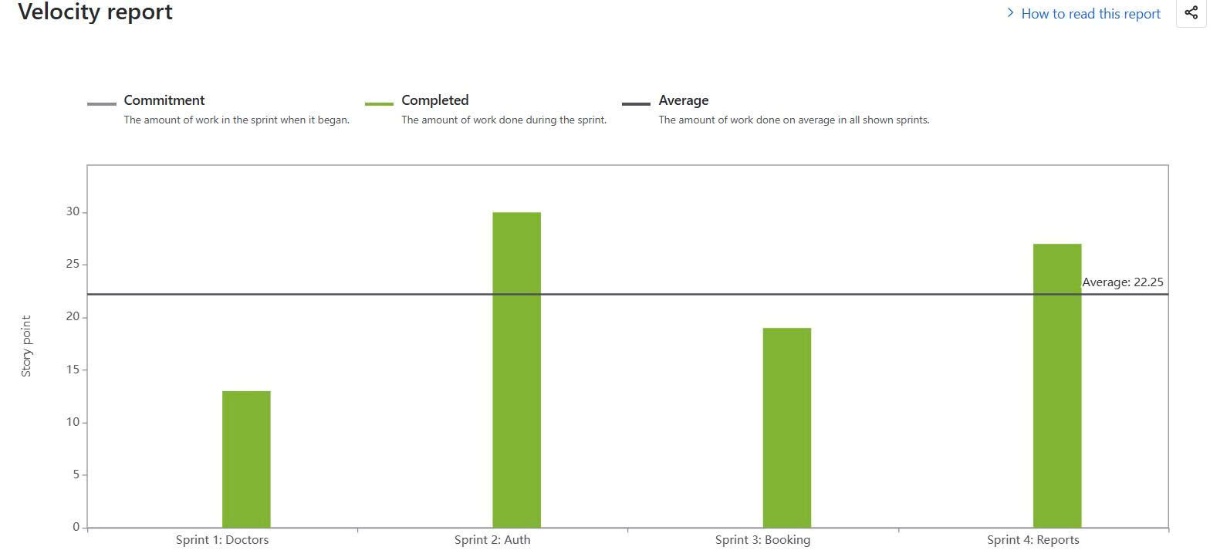
Mô tả Sprint: Booking

* Là một người dùng, tôi muốn hủy hoặc thay đổi lịch hẹn, nhờ đó tôi có thể điều chỉnh khi có thay đổi kế hoạch.
* Là một bác sĩ, tôi muốn xác nhận hoặc từ chối lịch hẹn, nhờ đó tôi có thể quản lý thời gian linh hoạt.
* Là một bác sĩ, tôi muốn cập nhật trạng thái lịch hẹn để theo dõi tiến trình.

1. **Report**

Là một admin, tôi muốn xem báo cáo thống kê để theo dõi hoạt động hệ thống.

Hiệu suất làm việc của nhóm được thể hiện qua biểu đồ Velocity sau:



Hình 5.4 Biểu đồ Velocity của các Sprint, thể hiện năng suất hoàn thành công việc

## Phân công

Bảng 5.1 Bảng phân công công việc

|  |  |  |
| --- | --- | --- |
| **STT** | **Sinh viên thực hiện** | **Công việc** |
| 1 | Nguyễn Hoài An  Kim Thạch Minh Trí | - Thiết kế giao diện bằng Figma.  - Xây dựng giao diện bằng React và Tailwind CSS. |
| 2 | Nguyễn Hoài An  Nguyễn Văn Tổng | - Xử lý các chức năng: đăng ký, đăng nhập, đặt lịch khám, xem danh sách bác sĩ. |
| 3 | Nguyễn Văn Tổng  Kim Thạch Minh Trí | - Xây dựng RESTful API với Node.js và Express.  - Xử lý logic: đặt lịch khám, xác nhận lịch hẹn, quản lý hồ sơ bác sĩ.  - Quản lý cơ sở dữ liệu MySQL. |
| 4 | Tất cả thành viên | - Quản lý tiến độ và backlog bằng Jira.  - Tổ chức kiểm thử API với Postman.  - Thiết lập Docker, CI/CD với GitHub Actions. |
| 5 | Nguyễn Hoài An | Viết báo cáo.  Thiết kế slide báo cáo. |

# KIỂM THỬ

## Chiến lược kiểm thử và công cụ sử dụng

### Các chiến lược kiểm thử

1. **Unit Tests**: Kiểm thử từng thành phần/chức năng riêng lẻ
2. **Integration Tests**: Kiểm thử tích hợp API và kết nối cơ sở dữ liệu
3. **End-to-End Tests**: Kiểm thử quy trình người dùng hoàn chỉnh
4. **Security Tests**: Quét lỗ hổng bảo mật và kiểm tra thông tin nhạy cảm
5. **Performance Tests**: Kiểm thử hiệu suất và tối ưu hóa
6. **Code Quality**: Phân tích tĩnh và kiểm tra chất lượng mã nguồn

### Mục tiêu kiểm thử

1. Đảm bảo tính chính xác của các chức năng nghiệp vụ
2. Phát hiện lỗi sớm trong quá trình phát triển
3. Đảm bảo tính bảo mật của hệ thống
4. Tối ưu hóa hiệu suất ứng dụng
5. Duy trì chất lượng mã nguồn cao

### Cấu trúc kiểm thử

#### Kiểm thử cấu trúc Frontend (React/Next.js)

frontend/

├── \_\_tests\_\_/ # Thư mục chứa các test cases

│ └── unit/ # Unit tests

│ ├── components/ # Kiểm thử các React components

│ │ └── Button.test.tsx # Kiểm thử component Button

│ └── utils/ # Kiểm thử các hàm tiện ích

│ └── auth.test.ts # Kiểm thử xác thực người dùng

├── jest.setup.js # Cấu hình Jest framework

├── coverage/ # Báo cáo độ bao phủ kiểm thử

└── e2e/ # End-to-end tests

└── login.spec.ts # Kiểm thử quy trình đăng nhập

Công nghệ sử dụng:

***Jest***: Framework kiểm thử chính

***React Testing Library***: Kiểm thử React components

***@testing-library/jest-dom***: Thư viện hỗ trợ kiểm thử DOM

***Playwright***: Kiểm thử end-to-end

#### Kiểm thử cấu trúc Backend (Node.js/Express)

backend/

├── \_\_tests\_\_/ # Thư mục chứa các test cases

│ └── unit/ # Unit tests

│ ├── models/ # Kiểm thử các model cơ sở dữ liệu

│ │ └── User.test.js # Kiểm thử User model

│ └── utils/ # Kiểm thử các hàm tiện ích

│ └── auth.test.js # Kiểm thử logic xác thực

├── test.js # Bộ kiểm thử chính

├── jest.setup.js # Cấu hình Jest

└── coverage/ # Báo cáo độ bao phủ kiểm thử

Công nghệ sử dụng:

***Jest***: Framework kiểm thử chính

***Supertest***: Kiểm thử HTTP endpoints

***Mock functions***: Mô phỏng cơ sở dữ liệu và dịch vụ bên ngoài

Công cụ kiểm thử chính:

* Postman: dùng để kiểm tra thủ công các API REST (POST, GET, PUT, DELETE).
* GitHub Actions (CI/CD - pipeline): phục vụ kiểm thử tự động khi push code.

## Kết quả kiểm thử

### Kiểm thử bằng Postman

#### Đăng nhập

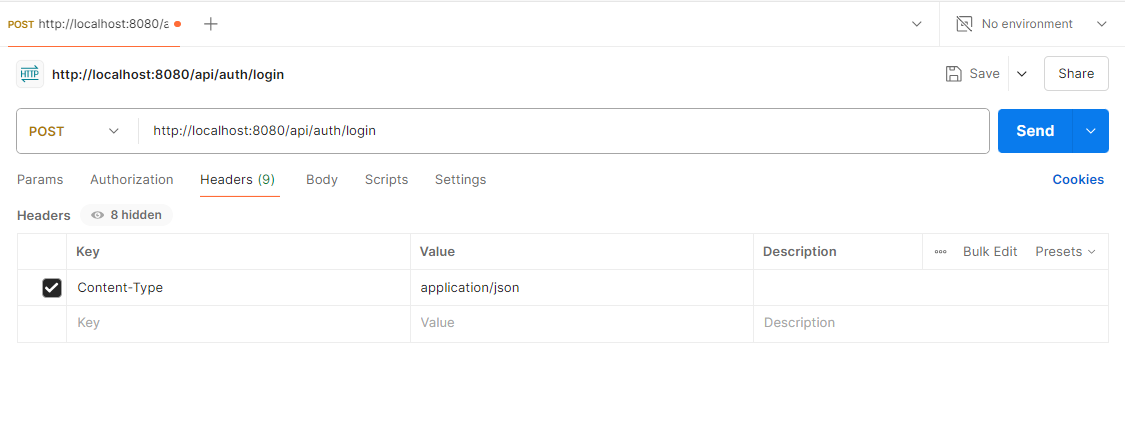
Gửi yêu cầu POST đến API đăng nhập:

Endpoint: http://localhost:8080/api/auth/login

Phương thức: POST.

Ở tab Headers ta điền các giá trị như sau:

* ***Key***: Content-Type
* ***Value***: application/json



Hình 6.1 Cấu hình Header trong Postman

Ở tab Body: chọn vào mục raw JSON, sau đó ta tiến hành gửi dữ liệu đăng nhập (email và mật khẩu) ở phần Body dưới định dạng JSON như sau:

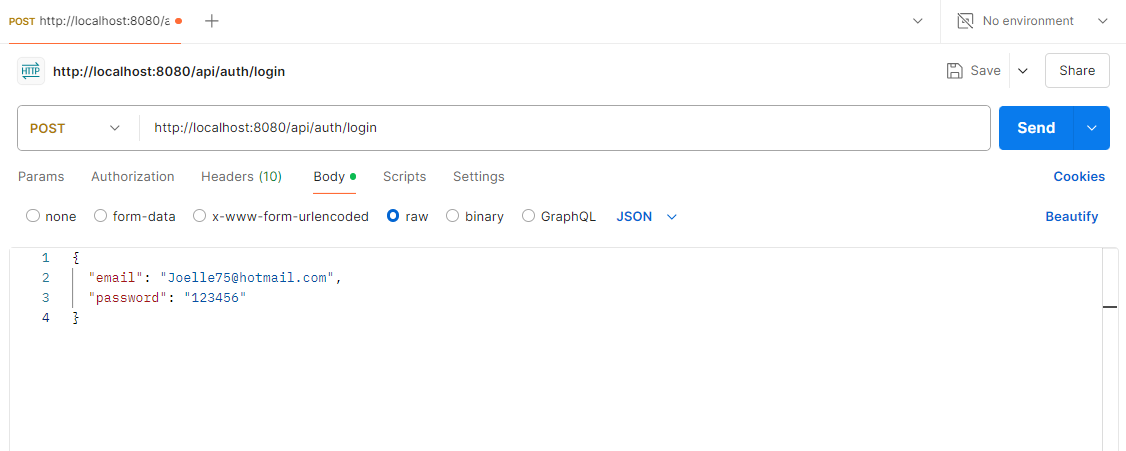
{

"email": "Joelle75@hotmail.com",

"password": "123456"

}

Sau khi cấu hình nội dung yêu cầu trong phần **Body** của Postman được thể hiện ở *Hình 6.2* dưới đây, sau đó nhấn Send.



Hình 6.2 Gửi yêu cầu đăng nhập với JSON Body

Sau khi gửi yêu cầu, hệ thống phản hồi với mã trạng thái 200 OK, kèm theo một đối tượng JSON chứa thông tin xác thực người dùng như **token**, **email**, **firstName**, **lastName** và vai trò (**Role**). Kết quả trả về được thể hiện trong *Hình 6.3*:



Hình 6.3 Phản hồi từ API đăng nhập – trả về token và thông tin người dùng

#### Đăng ký

Gửi yêu cầu POST đến API đăng ký:

http://localhost:8080/api/users/register

Phương thức: POST.

Ở tab Body: chọn vào mục raw JSON, sau đó ta tiến hành gửi dữ liệu đăng ký(email, password, name, role) ở phần Body dưới định dạng JSON như sau:

{

  "email": "patient2@example.com",

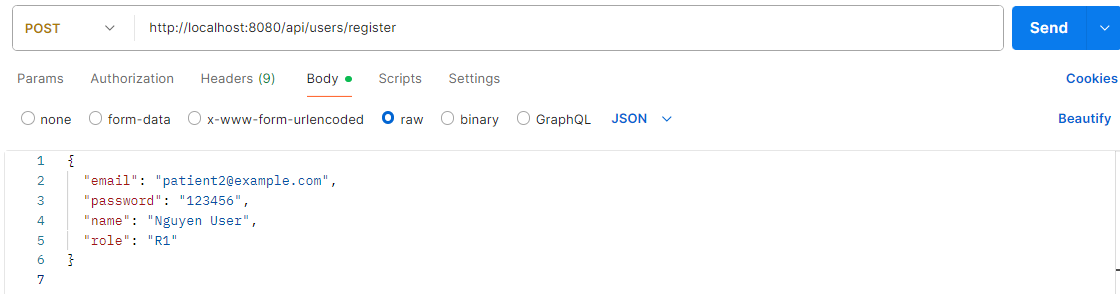
  "password": "123456",

  "name": "Nguyen User",

  "role": "R1"

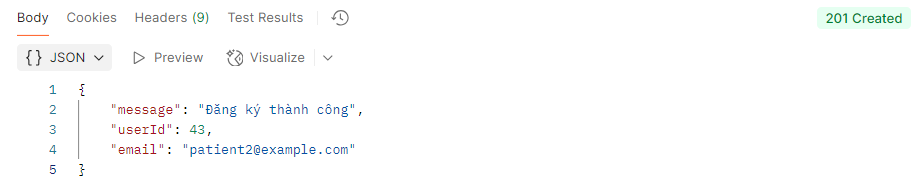
}

Sau khi đã thực hiện trình tự như trên, ta được như *Hình 6.4.*



Hình 6.4 Gửi yêu cầu đăng ký tài khoản

Sau đó nhấn Send. Một kết quả trả về 201 Created giống như hình 6.5 là bạn đã đăng ký thành công.



Hình 6.5 Kết quả trả về - đăng ký tài khoản

#### Đặt lịch

Sau khi login xong như *Hình 6.3*, ta sẽ sử dụng token đó để đặt lịch.

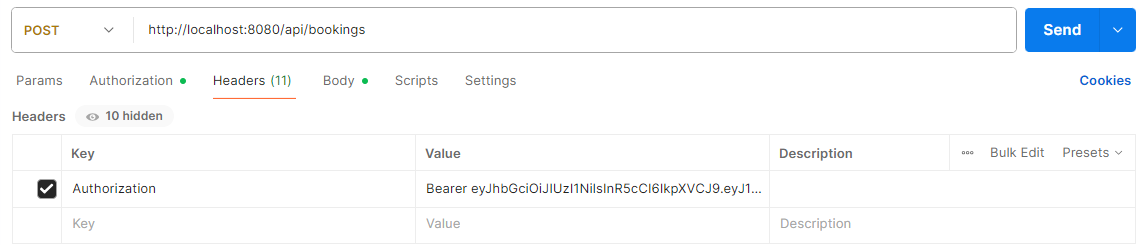
Ta thực hiện như sau: Đầu tiên truy cập endpoint:

<http://localhost:8080/api/bookings>

Tiếp theo ở tab Header điền như sau:

Key: Authorization

Value: Bearer ... (“...” ở đây là lấy token sau khi đã login), sau khi thực hiện ta sẽ được như Hình 6.6.



Hình 6.6 Gửi yêu cầu đặt lịch với token xác thực

Tiếp theo chuyển sang tab Body: chọn raw và JSON, sau đó điền vào các thông tin như sau:

{

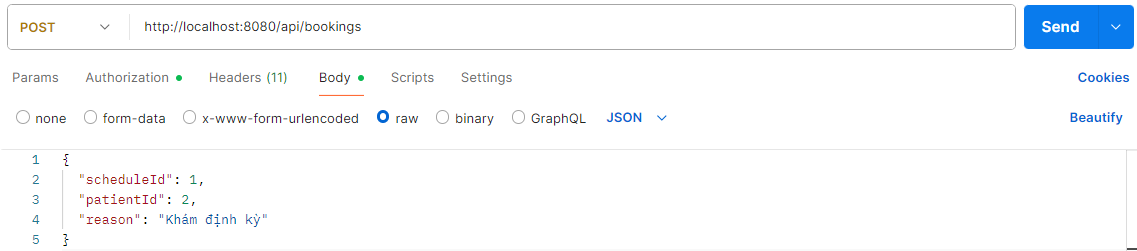
  "scheduleId": 1,

  "patientId": 2,

  "reason": "Khám định kỳ"

}

Ta được như *Hình 6.7*. Sau đó nhấn Send.



Hình 6.7 Gửi dữ liệu đặt lịch dưới dạng JSON

Kết quả trả về sẽ được như *Hình 6.8* bên dưới.



Hình 6.8 Kết quả trả về khi đặt lịch thành công từ Postman

Giải thích:

* + success: true: Đặt lịch thành công
  + id: ID của lịch hẹn vừa tạo
  + token: mã định danh (có thể dùng để xác nhận)
  + createdAt: thời điểm tạo lịch hẹn

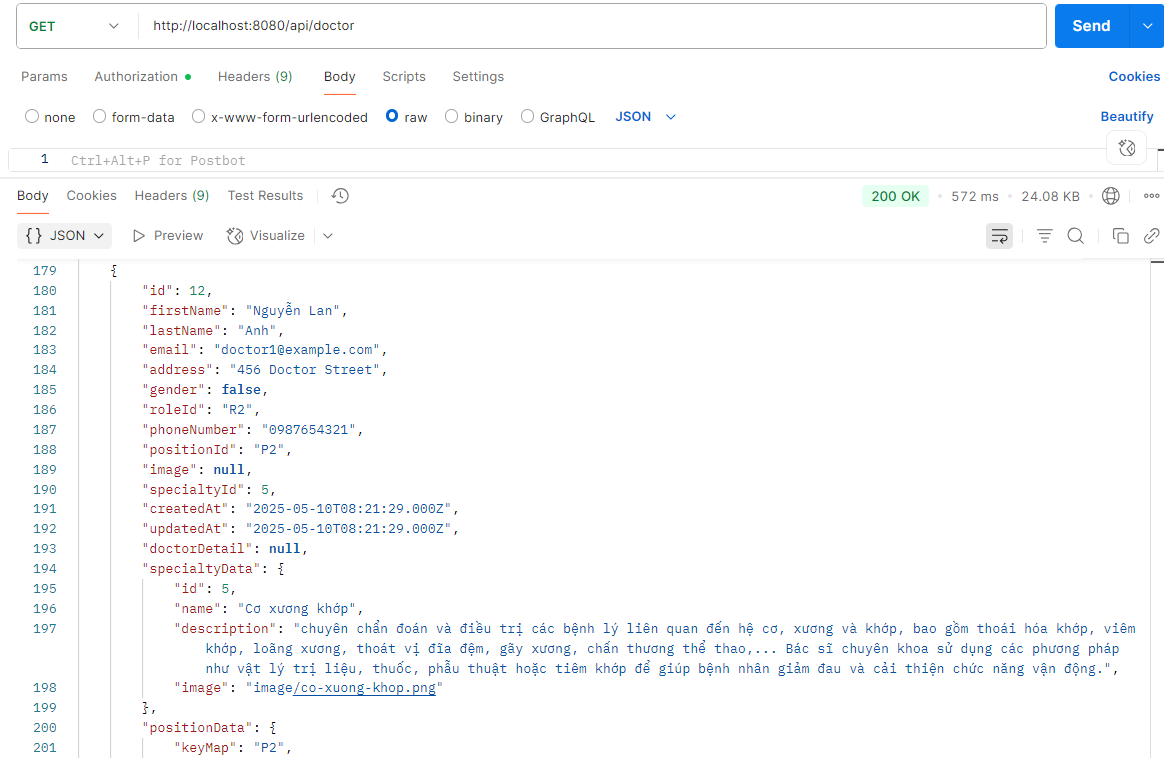
#### Lấy danh sách bác sĩ/chuyên khoa

1. **Lấy danh sách bác sĩ:**

Truy cập Endpoint:

http://localhost:8080/api/doctors

Phương thức: GET. Sau đó nhấn Send. Kết quả trả về là danh sách các bác sĩ.



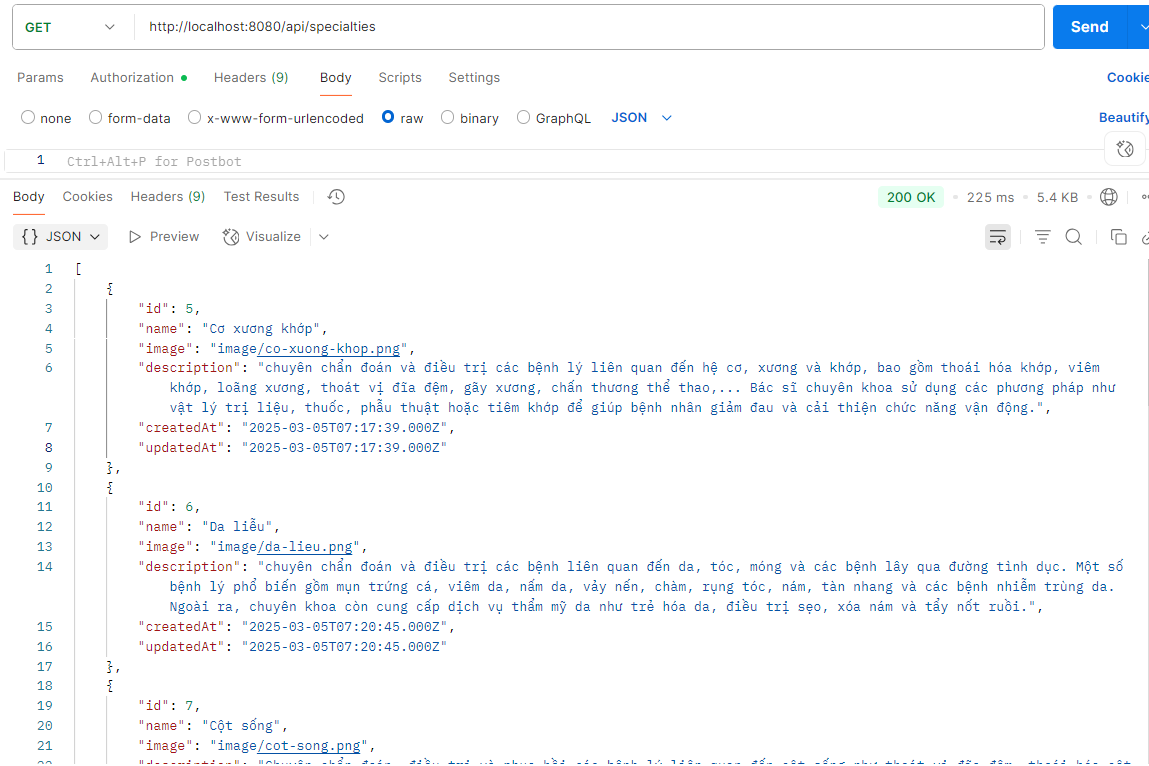
Hình 6.9 Kết quả trả về danh sách bác sĩ

1. **Lấy danh sách chuyên khoa:**

Truy cập Endpoint:

http://localhost:8080/api/specialties

Phương thức: GET. Sau đó nhấn Send. Kết quả trả về là danh sách các chuyên khoa dưới dạng JSON.



Hình 6.10 Kết quả trả về danh sách các chuyên khoa

### Kiểm thử tự động

Tổng quan:

* Tổng số test cases: 37 test cases
* Backend Tests: 26 test cases
* Frontend Tests: 11 test cases

#### Kiểm thử chức năng phía Backend:

* Kiểm thử xác thực người dùng: 8 tests
* Kiểm thử models cơ sở dữ liệu: 12 tests
* Kiểm thử tích hợp API: 6 tests

Test các cases chính:

1. Kiểm thử User Model:

* Tạo người dùng mới với thông tin hợp lệ
* Kiểm tra validation email và mật khẩu
* Cập nhật thông tin người dùng
* Xóa người dùng khỏi hệ thống
* Kiểm tra ràng buộc dữ liệu

2. Kiểm thử Authentication:

* Đăng nhập với thông tin đúng
* Xử lý đăng nhập thất bại
* Tạo và xác thực JWT token
* Mã hóa và xác minh mật khẩu
* Kiểm tra phân quyền người dùng

3. Kiểm thử API Integration:

* Các thao tác CRUD cho tất cả entities
* Xử lý lỗi và exception handling
* Kiểm tra định dạng response
* Validation dữ liệu đầu vào
* Kiểm tra status codes

#### Kiểm thử chức năng phía Frontend:

Kiểm thử React components: 7 tests

Kiểm thử hàm tiện ích: 4 tests

1. Kiểm thử Components:

* + Render đúng nội dung component
  + Xử lý tương tác người dùng
  + Validation form và input
  + Truyền props giữa components
  + Xử lý state management

2. Kiểm thử Utilities:

* + Các hàm hỗ trợ xác thực
  + Định dạng dữ liệu hiển thị
  + Gọi API và xử lý response
  + Xử lý lỗi và loading states

#### Tự động hóa và CI/CD

1. Scripts Kiểm thử:

Hệ thống cung cấp các lệnh kiểm thử sau:

# Chạy tất cả kiểm thử

npm run test:all

# Chạy kiểm thử riêng lẻ

npm run test:frontend # Chỉ kiểm thử Frontend

npm run test:backend # Chỉ kiểm thử Backend

# Kiểm thử trước khi commit

npm run pre-commit # Kiểm thử các file đã thay đổi

# Kiểm thử cục bộ

npm run test:local # Kiểm thử cơ bản trên máy local

1. GitHub Actions CI/CD Pipeline
2. Quy trình tự động:
   * Trigger: Tự động chạy khi push code hoặc tạo Pull Request
   * Setup Environment: Cài đặt Node.js và dependencies
   * Run Tests: Chạy tất cả bộ kiểm thử
   * Coverage Report: Tạo báo cáo độ bao phủ
   * Security Scan: Quét lỗ hổng bảo mật
   * Build: Tạo Docker images nếu tests thành công
   * Deploy: Triển khai lên môi trường staging/production
3. Cấu hình Pipeline:
   * Môi trường: Ubuntu latest, Node.js 18+
   * Timeout: 10 phút cho toàn bộ pipeline
   * Retry: Tự động retry 2 lần nếu thất bại
   * Notification: Thông báo kết quả qua email/Slack
4. Công cụ và Framework
5. Testing Frameworks
   * Jest: Framework kiểm thử chính cho cả frontend và backend
   * React Testing Library: Chuyên dụng cho kiểm thử React components
   * Supertest: Kiểm thử API endpoints và HTTP requests
   * Playwright: Kiểm thử end-to-end (đang triển khai)
6. Công cụ Chất lượng Code
   * ESLint: Kiểm tra lỗi cú pháp và coding standards
   * Prettier: Định dạng code tự động
   * Husky: Git hooks cho pre-commit testing
   * Jest Coverage: Báo cáo độ bao phủ kiểm thử
7. Môi trường Kiểm thử
   * Local Development: Kiểm thử trên máy phát triển
   * Docker Environment: Kiểm thử trong container
   * CI/CD Environment: Kiểm thử tự động trên GitHub Actions
   * Staging Environment: Kiểm thử trước khi production

Kết quả:

* + 37 test cases bao phủ các chức năng quan trọng
  + CI/CD pipeline tự động với GitHub Actions
  + Nhiều cấp độ kiểm thử từ unit đến integration tests
  + Quality gates đảm bảo chất lượng trước khi deploy

# ĐÁNH GIÁ VÀ KẾT LUẬN

## Đánh giá

Đề tài "Xây dựng Website đặt lịch khám bệnh trực tuyến" đã hoàn thành đúng theo tiến độ, đảm bảo đầy đủ các chức năng theo yêu cầu và tuân thủ quy trình phát triển phần mềm. Trong quá trình thực hiện, nhóm cũng đã gặp phải một số khó khăn:

**Triển khai Docker và CI/CD:** Việc cấu hình Docker Compose và thiết lập GitHub Actions ban đầu gặp nhiều lỗi liên quan đến môi trường và biến hệ thống.

**Tích hợp giao diện với API:** Trong quá trình kết nối giữa frontend và backend, một số lỗi như sai định dạng dữ liệu, lỗi phân quyền, hoặc lỗi CORS thường xuyên xảy ra, cần nhiều thời gian debug và kiểm thử.

**Khó khăn về quản lý thời gian**: Các thành viên đều có lịch học và công việc cá nhân khác nhau, nên việc phân phối thời gian làm việc nhóm ban đầu chưa hiệu quả. Chúng tôi đã cải thiện điều này bằng cách áp dụng quản lý tiến độ theo sprint trên Jira và chia công việc nhỏ rõ ràng hơn.

## Kết luận

### Bài học rút ra

Kỹ năng làm việc nhóm và phân công nhiệm vụ: Mỗi thành viên đều học được cách làm việc theo nhóm hiệu quả hơn, biết cách phối hợp, hỗ trợ lẫn nhau và chủ động theo dõi tiến độ chung. Sử dụng công cụ chuyên nghiệp trong phát triển phần mềm:

* Jira giúp nhóm kiểm soát tiến độ và công việc.
* Postman giúp kiểm thử API dễ dàng hơn.
* GitHub Actions giúp tự động hóa quá trình kiểm thử và triển khai.
* Docker giúp dễ dàng thiết lập môi trường đồng nhất giữa các thành viên.

### Đề xuất cải tiến trong tương lai

Dưới đây là một số tính năng đề xuất sẽ được phát triển trong tương lai:

* **Tích hợp thanh toán online:** Cho phép người dùng thanh toán phí khám bệnh ngay khi đặt lịch.
* **Gửi thông báo nhắc lịch qua email/SMS:** Hệ thống sẽ tự động nhắc người dùng và bác sĩ trước lịch hẹn để giảm tình trạng quên lịch.

# PHỤ LỤC

**Hướng dẫn cài đặt**

1. ***Cài đặt Local (Development)***

1. Clone repository:

git clone https://github.com/TongNguyenvk/CNPM\_WebSiteDKKhamBenh.git

cd CNPM\_WebSiteDKKhamBenh

2. Cài đặt dependencies cho Backend:

cd backend

npm install

3. Cài đặt dependencies cho Frontend:

cd ../frontend

npm install

4. Thiết lập cơ sở dữ liệu MySQL:

CREATE DATABASE cnpm\_hospital\_booking;

5. Chạy migrations và seeders:

cd backend

npm run migrate

npm run seed

1. ***Cài đặt với Docker (Khuyến nghị)***

1. Đảm bảo đã cài đặt Docker và Docker Compose

2. Chạy toàn bộ ứng dụng:

docker-compose up -d

3. Kiểm tra trạng thái containers:

docker-compose ps

1. ***Cài đặt nhanh (Quick Start)***

Clone và cài đặt

git clone https://github.com/TongNguyenvk/CNPM\_WebSiteDKKhamBenh.git

cd CNPM\_WebSiteDKKhamBenh

Chạy với Docker

docker-compose up -d

Hoặc chạy local

npm run dev

*Cấu hình*

Biến môi trường Backend

Tạo file `.env` trong thư mục `backend`:

# Server Configuration

PORT=8080

NODE\_ENV=development

# Database Configuration

DB\_HOST=localhost

DB\_PORT=3306

DB\_NAME=cnpm\_hospital\_booking

DB\_USER=root

DB\_PASSWORD=your\_mysql\_password

# JWT Configuration

JWT\_SECRET=your\_super\_secret\_jwt\_key\_here

JWT\_EXPIRES\_IN=7d

# Email Configuration (optional)

EMAIL\_HOST=smtp.gmail.com

EMAIL\_PORT=587

EMAIL\_USER=your\_email@gmail.com

EMAIL\_PASS=your\_app\_password

# Upload Configuration

UPLOAD\_PATH=./uploads

MAX\_FILE\_SIZE=5MB

Biến môi trường Frontend

Tạo file `.env.local` trong thư mục `frontend`:

# API Configuration

NEXT\_PUBLIC\_API\_URL=http://localhost:8080/api

NEXT\_PUBLIC\_UPLOAD\_URL=http://localhost:8080/uploads

# App Configuration

NEXT\_PUBLIC\_APP\_NAME="Website Đăng Ký Khám Bệnh"

NEXT\_PUBLIC\_APP\_VERSION=1.0.0

Cấu hình Database

1. Tạo database MySQL:

CREATE DATABASE cnpm\_hospital\_booking CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

2. Import dữ liệu mẫu:

cd backend

npm run migrate # Chạy migrations

npm run seed # Import dữ liệu mẫu

1. ***Chạy ứng dụng***

Development Mode

Cách 1: Chạy đồng thời Frontend + Backend

npm run dev

Cách 2: Chạy riêng từng service

# Terminal 1 - Backend

npm run dev:backend

# Terminal 2 - Frontend

npm run dev:frontend

Truy cập ứng dụng:

- Frontend: http://localhost:3000

- Backend API: http://localhost:8080

- API Documentation: http://localhost:8080/api-docs

Docker Mode

Chạy toàn bộ stack:

npm run docker:up

Các lệnh Docker hữu ích:

npm run docker:down # Dừng containers

npm run docker:logs # Xem logs

npm run docker:clean # Dọn dẹp containers và images

Testing Mode

npm run test # Chạy tất cả tests

npm run test:frontend # Test frontend only

npm run test:backend # Test backend only

Liên kết GitHub: <https://github.com/TongNguyenvk/CNPM_WebSiteDKKhamBenh>