

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO BÀI TẬP LỚN

Môn học	Kiến trúc và thiết kế phần mềm
Họ và tên	Phạm Đức Chính
Mã sinh viên	B21DCCN181
Lớp	CNPM01
Nhóm lớp	03
Giảng viên	Thầy Trần Đình Quế

Hà Nội – 2025

CHAPTER 1: REQUIREMENTS OF E-COMMERCE

1.1 Xác định yêu cầu

Giới thiệu hệ thống

Hệ thống Health-Care là một nền tảng phần mềm được xây dựng nhằm số hóa và tối ưu hóa quy trình khám chữa bệnh trong các cơ sở y tế. Hệ thống hỗ trợ tương tác hiệu quả giữa các đối tượng như bệnh nhân, bác sĩ, y tá, dược sĩ, kỹ thuật viên xét nghiệm, nhà cung cấp bảo hiểm và quản trị viên hệ thống.

Thông qua hệ thống, bệnh nhân có thể đặt lịch khám, xem hồ sơ y tế, nhận kết quả xét nghiệm và đơn thuốc điện tử. Bác sĩ và y tá có thể dễ dàng cập nhật thông tin điều trị, quản lý lịch làm việc và hỗ trợ bệnh nhân. Ngoài ra, hệ thống còn giúp tự động hóa quy trình xác nhận bảo hiểm, theo dõi thuốc, và xử lý kết quả xét nghiệm.

Mục tiêu của hệ thống là nâng cao chất lượng chăm sóc sức khỏe, tiết kiệm thời gian, giảm thiểu sai sót và mang lại trải nghiệm tốt nhất cho người dùng.

1.1.1 Các tác nhân (Actors)

Dưới đây là các tác nhân chính trong hệ thống cùng với mô tả chi tiết vai trò và ảnh hưởng của họ đến hệ thống:

Tác nhân	Mô tả
Bệnh nhân (Patient)	Là người sử dụng chính của hệ thống. Có thể đăng ký tài khoản, đăng nhập, đặt lịch khám với bác sĩ, xem lịch sử khám chữa bệnh, nhận đơn thuốc, kết quả xét nghiệm và cập nhật thông tin cá nhân. Bệnh nhân là người khởi tạo hầu hết các quy trình trong hệ thống.
Bác sĩ (Doctor)	Quản lý lịch hẹn, chẩn đoán bệnh, kê đơn thuốc, cập nhật hồ sơ y tế của bệnh nhân. Ngoài ra, bác sĩ còn đọc kết quả xét nghiệm từ phòng lab và có thể giới thiệu bệnh nhân đến bác sĩ chuyên khoa. Họ đóng vai trò trung tâm trong việc điều trị và theo dõi sức khỏe bệnh nhân.
Y tá (Nurse)	Hỗ trợ bác sĩ trong quá trình điều trị: đo và ghi nhận các chỉ số sinh tồn (huyết áp, nhiệt độ, mạch...), chuẩn bị hồ sơ bệnh nhân, chăm sóc

Tác nhân	Mô tả
	bệnh nhân sau điều trị và thực hiện chỉ định từ bác sĩ. Y tá là người thường xuyên cập nhật tình trạng bệnh nhân trên hệ thống.
Quản trị viên (Administrator)	Quản lý người dùng hệ thống, phân quyền truy cập, thiết lập và quản lý lịch làm việc của bác sĩ, xử lý lỗi hệ thống và đảm bảo bảo mật thông tin. Họ giúp vận hành hệ thống một cách trơn tru.
Dược sĩ (Pharmacist)	Kiểm tra đơn thuốc từ bác sĩ, cấp phát thuốc cho bệnh nhân, cập nhật tồn kho thuốc và theo dõi hạn sử dụng thuốc. Dược sĩ đảm bảo việc cấp phát thuốc đúng người, đúng loại và đúng liều.
Nhà cung cấp bảo hiểm (Insurance Provider)	Xác minh thông tin bảo hiểm của bệnh nhân, xử lý yêu cầu thanh toán và cập nhật trạng thái chi trả. Họ giúp đảm bảo quyền lợi của bệnh nhân và hỗ trợ quy trình tài chính.
Kỹ thuật viên xét nghiệm (Laboratory Technician)	Thực hiện các xét nghiệm y tế theo yêu cầu của bác sĩ, nhập kết quả xét nghiệm lên hệ thống và trao đổi với bác sĩ để hỗ trợ quá trình chẩn đoán. Họ là cầu nối giữa quá trình kiểm tra và điều trị.

1.1.2 Chức năng theo từng tác nhân

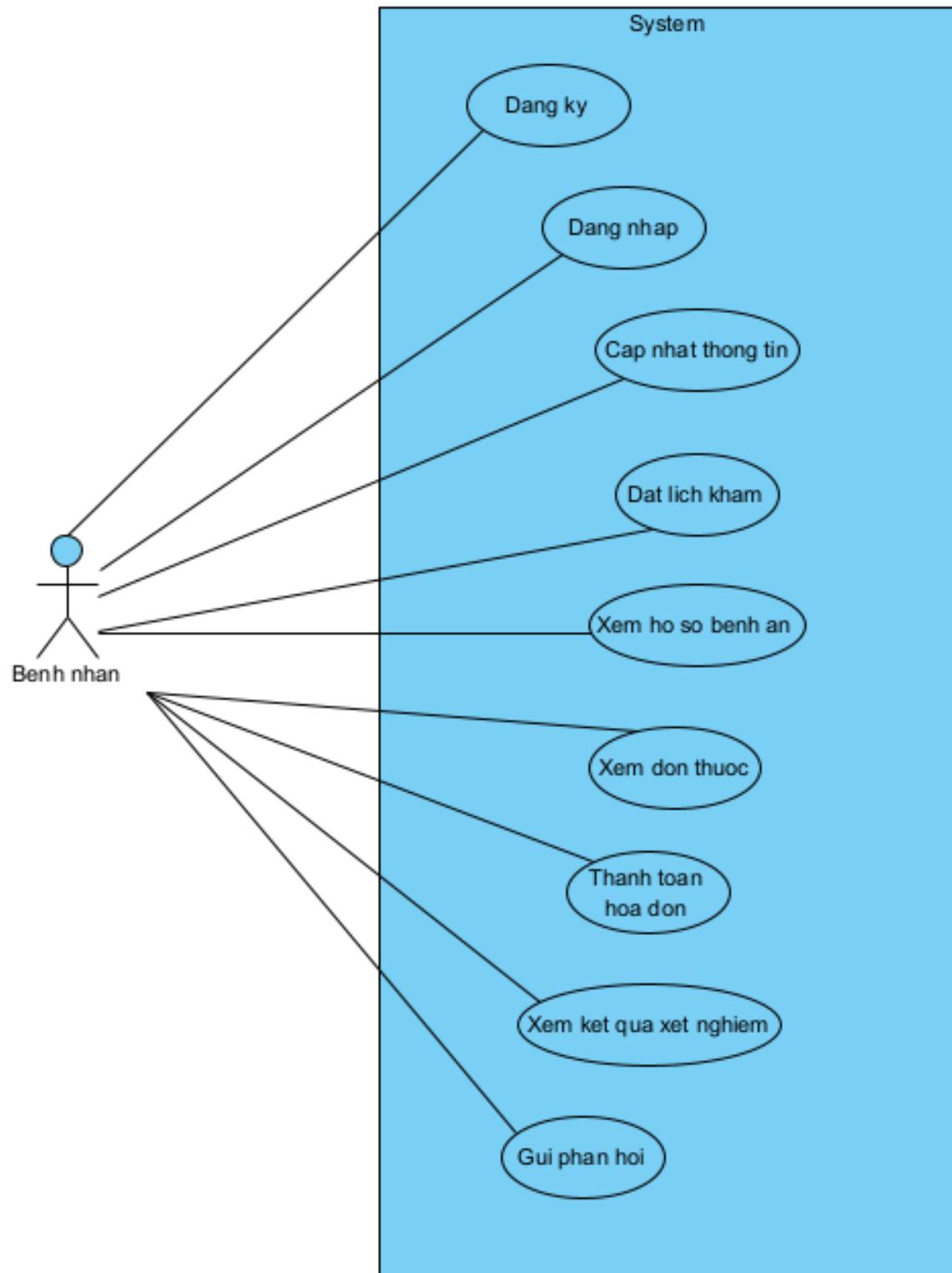
Tác nhân (Actor)	Chức năng (Use Cases)
1. Bệnh nhân (Patient)	<ul style="list-style-type: none"> - Đăng ký tài khoản / Đăng nhập bằng OTP/email/pass - Cập nhật thông tin cá nhân, số BHYT, người thân liên hệ - Tìm kiếm và chọn bác sĩ theo chuyên khoa, lịch rảnh - Đặt lịch khám (khám tại viện / khám online) - Nhận thông báo xác nhận lịch khám và nhắc lịch tự động - Xem hồ sơ bệnh án (theo từng lần khám) - Xem và tải kết quả xét nghiệm (PDF, ảnh) - Nhận đơn thuốc và chỉ định từ bác sĩ - Gửi câu hỏi/tư vấn bác sĩ sau khám - Xem hóa đơn và thanh toán dịch vụ - Theo dõi trạng thái yêu cầu bảo hiểm (nếu có)

Tác nhân (Actor)	Chức năng (Use Cases)
2. Bác sĩ (Doctor)	<ul style="list-style-type: none"> - Xem lịch làm việc và các ca khám hôm nay - Quản lý danh sách bệnh nhân theo lịch hẹn - Xem hồ sơ bệnh án trước đó của bệnh nhân - Ghi chú lâm sàng, chẩn đoán bệnh - Lập đơn thuốc và chỉ định cận lâm sàng (xét nghiệm, chụp chiếu) - Nhận kết quả xét nghiệm và cập nhật chẩn đoán - Gửi phản hồi hoặc tư vấn sau khám cho bệnh nhân - Kết thúc hồ sơ khám và tạo báo cáo bệnh án điện tử
3. Y tá (Nurse)	<ul style="list-style-type: none"> - Tiếp nhận bệnh nhân tại phòng khám hoặc từ hệ thống - Đo chỉ số sinh tồn (nhiệt độ, huyết áp, mạch, SPO2...) và cập nhật hồ sơ - Hỗ trợ bác sĩ nhập dữ liệu hồ sơ khám bệnh - Theo dõi bệnh nhân đang nằm viện (nếu có phân hệ điều trị nội trú) - Thông báo cho bác sĩ khi có bất thường - Hướng dẫn bệnh nhân về đơn thuốc và lịch tái khám
4. Quản trị viên (Administrator)	<ul style="list-style-type: none"> - Quản lý tài khoản người dùng: bệnh nhân, bác sĩ, y tá, dược sĩ,... - Phân quyền truy cập hệ thống - Quản lý lịch làm việc của bác sĩ, sắp xếp lịch hẹn - Giám sát hoạt động hệ thống và nhật ký truy cập (log) - Cập nhật thông tin chuyên khoa, phòng khám, dịch vụ - Quản lý hệ thống email/sms thông báo - Xử lý các lỗi kỹ thuật hoặc yêu cầu hỗ trợ từ người dùng
5. Dược sĩ (Pharmacist)	<ul style="list-style-type: none"> - Xem đơn thuốc từ bác sĩ - Xác nhận cấp phát thuốc hoặc báo hết hàng - Quản lý tồn kho thuốc: nhập/xuất/tồn - Cập nhật lô thuốc, hạn sử dụng

Tác nhân (Actor)	Chức năng (Use Cases)
	<ul style="list-style-type: none"> - Tạo danh sách thuốc thay thế (generic) khi cần - In phiếu thuốc hoặc xuất hóa đơn cấp phát
6. Kỹ thuật viên xét nghiệm (Laboratory Technician)	<ul style="list-style-type: none"> - Nhận chỉ định xét nghiệm từ bác sĩ - Thực hiện xét nghiệm (máu, nước tiểu, chụp X-quang, siêu âm,...) - Nhập kết quả xét nghiệm lên hệ thống (có thể kèm hình ảnh, file đính kèm) - Gửi thông báo cho bác sĩ khi có kết quả - Lưu trữ kết quả trong hồ sơ bệnh nhân

1.1.3 Use Case Diagram

1.1.3.1 Use Case Diagram – Tác nhân: Bệnh nhân (Patient)

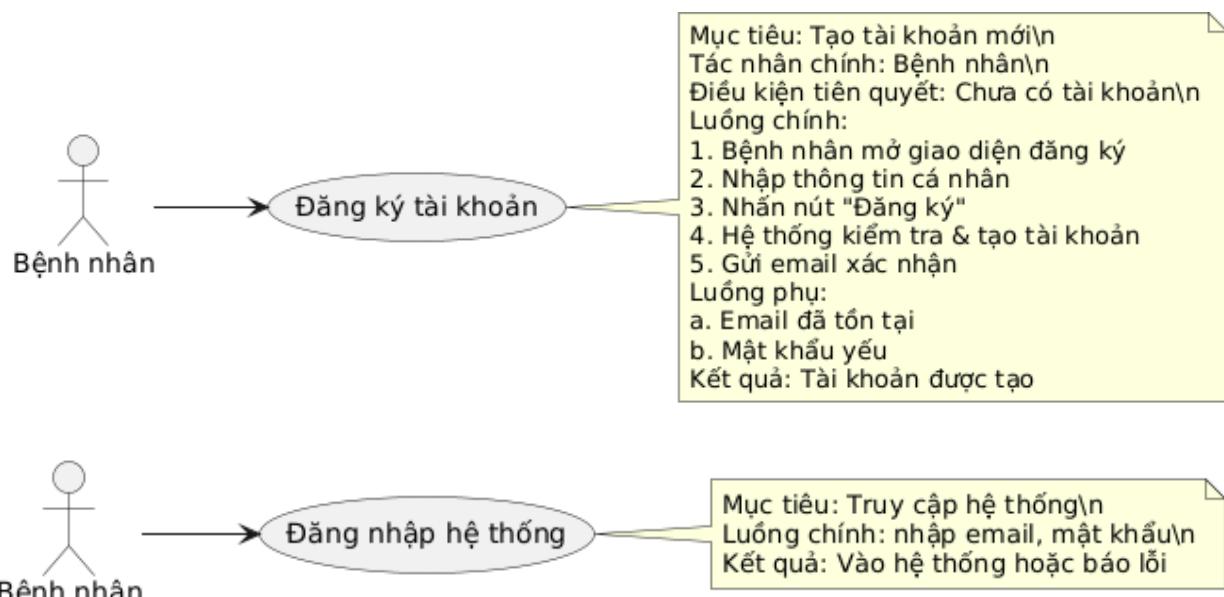


- Tác nhân: **Patient**
- Các Use Case:
 - Đăng ký/Đăng nhập

- Cập nhật thông tin cá nhân
 - Đặt lịch khám
 - Hủy lịch hẹn
 - Xem lịch khám
 - Xem hồ sơ bệnh án
 - Xem kết quả xét nghiệm
 - Xem đơn thuốc
 - Gửi phản hồi / tư vấn bác sĩ
 - Thanh toán hóa đơn
-

Phân tích từng Use Case – Bệnh nhân

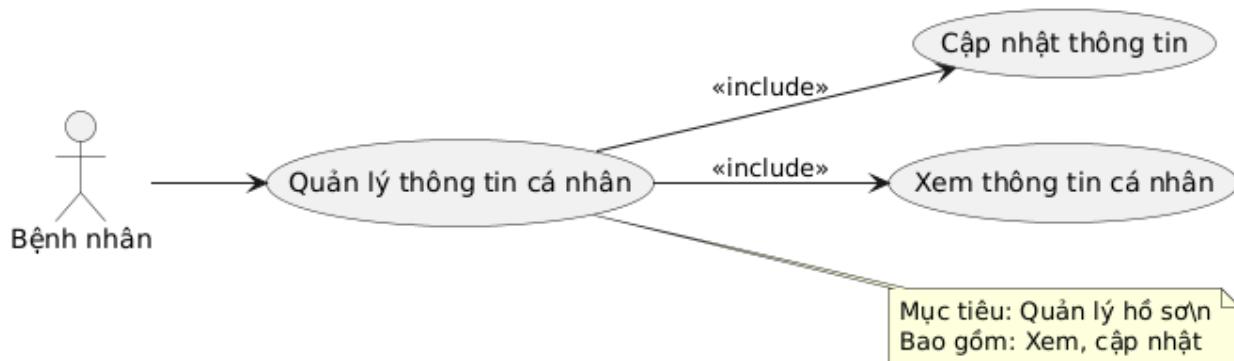
Use Case 1: Đăng ký / Đăng nhập



Mục	Nội dung
Tên Use Case	Đăng ký / Đăng nhập
Mô tả	Cho phép người dùng tạo tài khoản hoặc truy cập bằng OTP/email/mật khẩu
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Người dùng chưa có tài khoản (đối với đăng ký) hoặc đã có tài khoản (đối với đăng nhập)
Hậu điều kiện	Người dùng được chuyển đến trang chính (dashboard) của hệ thống

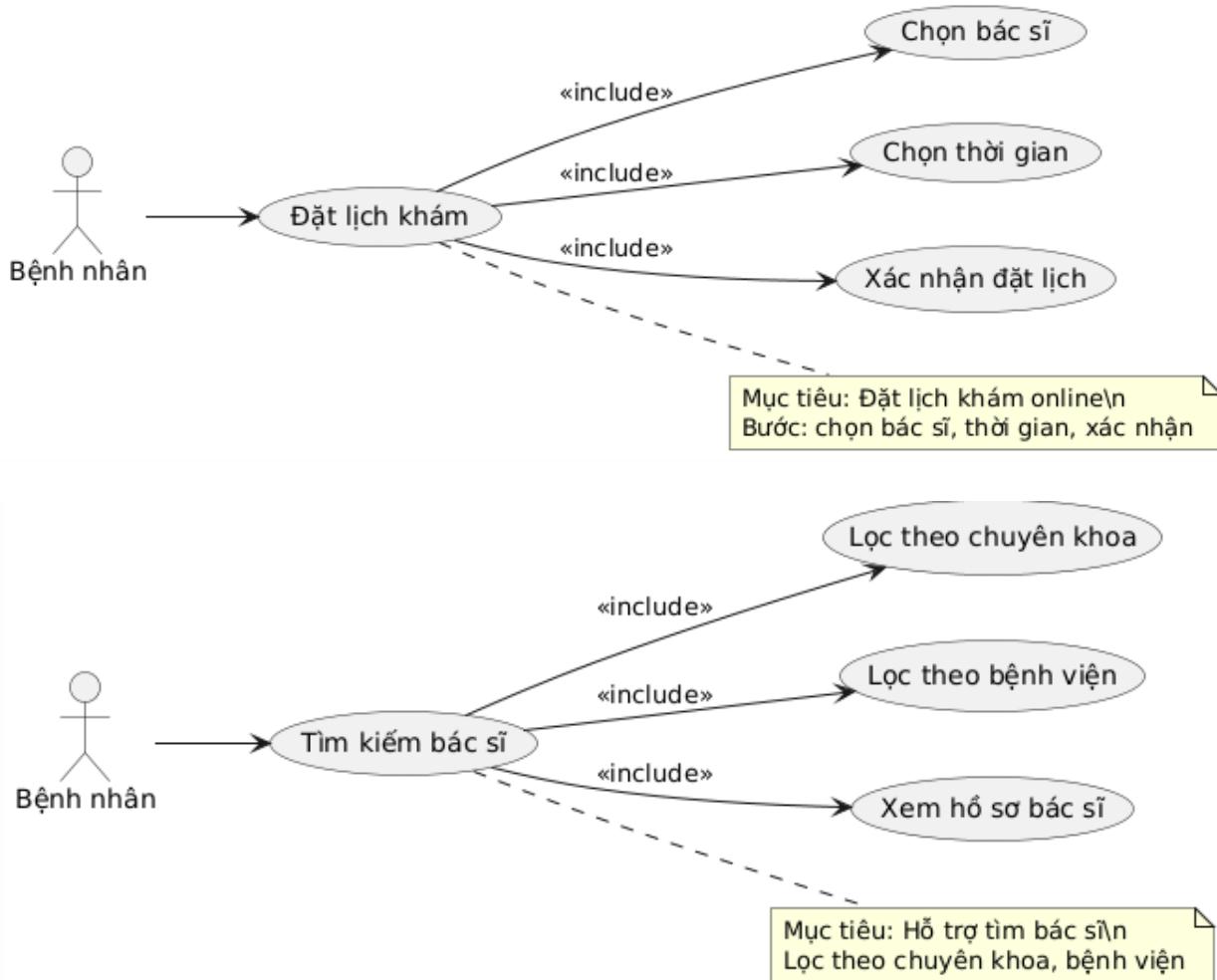
Mục	Nội dung
Luồng chính	1. Nhập số điện thoại/email 2. Nhập OTP hoặc mật khẩu 3. Xác thực thành công 4. Truy cập hệ thống
Luồng thay thế	- OTP sai hoặc hết hạn - Tài khoản không tồn tại - Cho phép đăng ký mới

Use Case 2: Cập nhật thông tin cá nhân



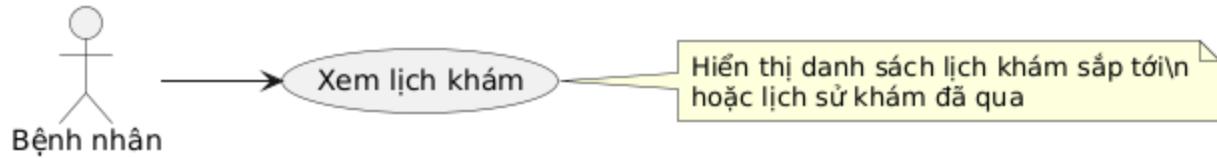
Mục	Nội dung
Tên Use Case	Cập nhật thông tin cá nhân
Mô tả	Cho phép người dùng cập nhật thông tin cá nhân, số BHYT, người thân liên hệ
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Đã đăng nhập
Hậu điều kiện	Thông tin được cập nhật thành công trong hệ thống
Luồng chính	1. Truy cập mục “Hồ sơ” 2. Cập nhật thông tin 3. Bấm “Lưu” 4. Hệ thống xác nhận cập nhật thành công
Luồng thay thế	Không kết nối mạng, lỗi lưu dữ liệu

Use Case 3: Đặt lịch khám



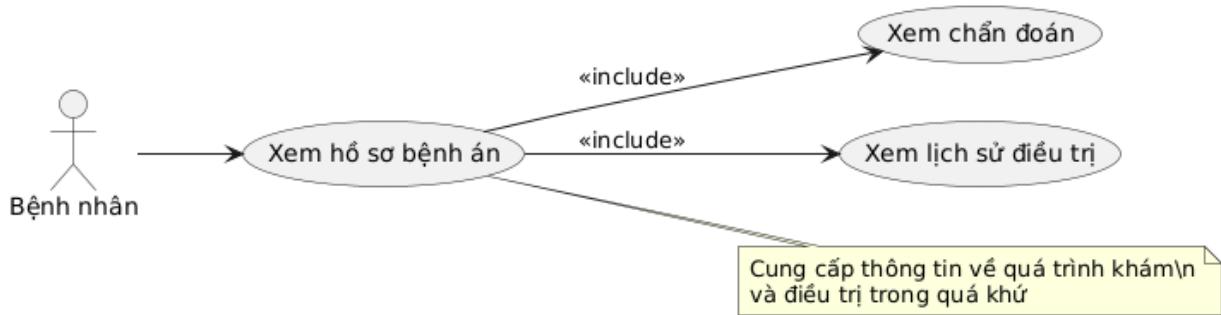
Mục	Nội dung
Tên Use Case	Đặt lịch khám
Mô tả	Cho phép bệnh nhân đặt lịch khám với bác sĩ theo chuyên khoa và khung giờ phù hợp
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Đã đăng nhập, đã cập nhật thông tin
Hậu điều kiện	Lịch hẹn được xác nhận và lưu lại trong hệ thống
Luồng chính	<ol style="list-style-type: none"> Chọn chuyên khoa → bác sĩ → ngày/giờ Nhập triệu chứng ban đầu (tùy chọn) Xác nhận lịch Nhận thông báo xác nhận
Luồng thay thế	Lịch bị trùng, bác sĩ không có mặt, yêu cầu bệnh nhân chọn lại

Use Case 4: Xem lịch khám



Mục	Nội dung
Tên Use Case	Xem lịch khám
Mô tả	Cho phép bệnh nhân xem danh sách các lịch hẹn sắp tới
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Đã đăng nhập
Hậu điều kiện	Hiển thị danh sách lịch khám
Luồng chính	1. Truy cập “Lịch khám” 2. Hệ thống hiển thị danh sách lịch sắp tới
Luồng thay thế	Không có lịch khám nào

Use Case 5: Xem hồ sơ bệnh án



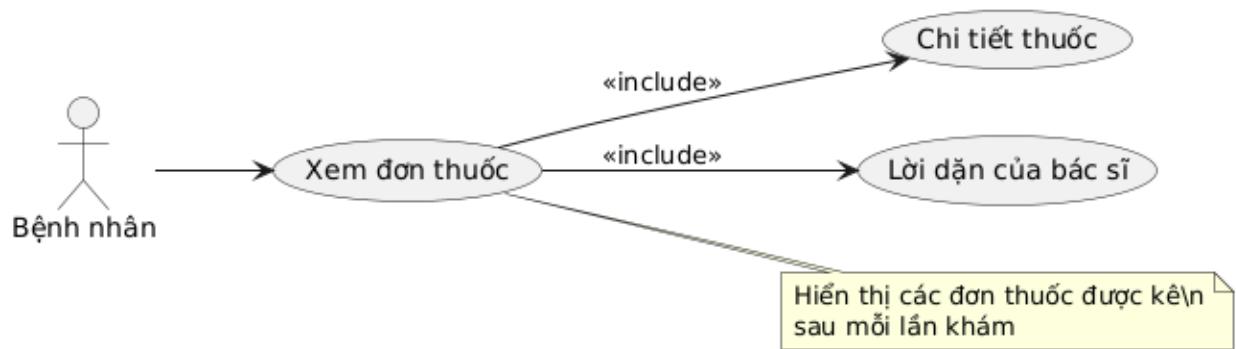
Mục	Nội dung
Tên Use Case	Xem hồ sơ bệnh án
Mô tả	Bệnh nhân có thể xem lại toàn bộ hồ sơ khám bệnh trước đó
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Đã từng khám bệnh
Hậu điều kiện	Hồ sơ hiển thị dưới dạng chi tiết hoặc PDF
Luồng chính	1. Truy cập mục “Hồ sơ khám bệnh” 2. Chọn thời gian / đợt khám 3. Xem chi tiết hoặc tải về
Luồng thay thế	Hồ sơ chưa được cập nhật, không tìm thấy dữ liệu

Use Case 6: Xem kết quả xét nghiệm



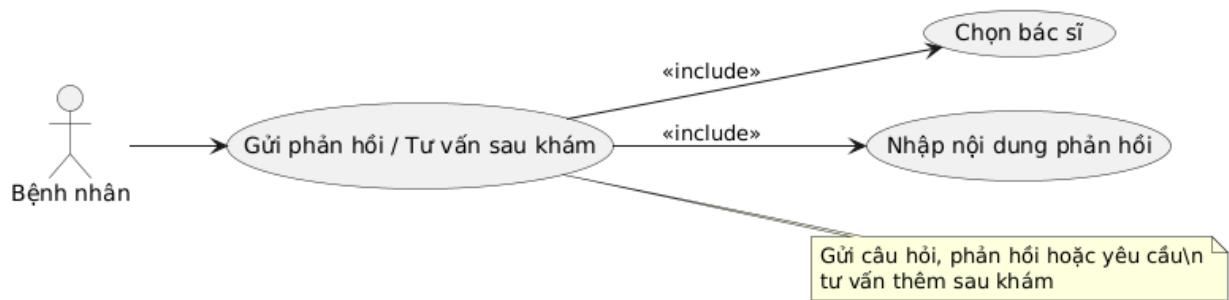
Mục	Nội dung
Tên Use Case	Xem kết quả xét nghiệm
Mô tả	Xem kết quả từ các xét nghiệm được chỉ định trong đợt khám
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Có chỉ định xét nghiệm và đã có kết quả từ phòng lab
Hậu điều kiện	Bệnh nhân có thể tải về kết quả hoặc in
Luồng chính	1. Vào mục “Xét nghiệm” 2. Chọn đợt khám 3. Xem kết quả (PDF, ảnh)
Luồng thay thế	Chưa có kết quả, lỗi hiển thị

Use Case 7: Xem đơn thuốc



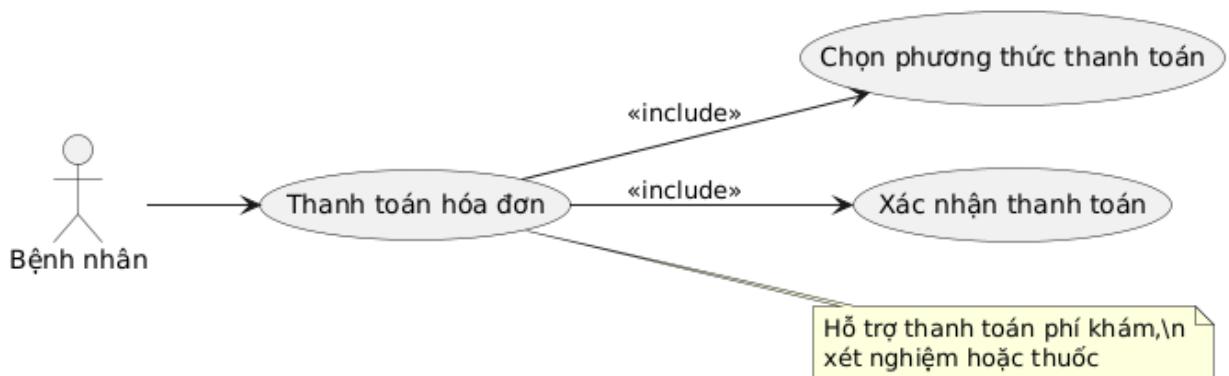
Mục	Nội dung
Tên Use Case	Xem đơn thuốc
Mô tả	Xem và tải đơn thuốc do bác sĩ kê sau khi khám
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Hồ sơ khám đã được kết thúc và có đơn thuốc
Hậu điều kiện	Bệnh nhân có thể mua thuốc hoặc gửi đơn đến nhà thuốc
Luồng chính	1. Truy cập mục “Đơn thuốc” 2. Xem hoặc tải về 3. Chọn nhà thuốc nhận đơn (nếu có)
Luồng thay thế	Không có đơn thuốc, lỗi hiển thị

Use Case 8: Gửi phản hồi / tư vấn sau khám



Mục	Nội dung
Tên Use Case	Gửi phản hồi / hỏi bác sĩ
Mô tả	Gửi phản hồi, thắc mắc hoặc yêu cầu tư vấn bổ sung sau khi khám
Tác nhân liên quan	Bệnh nhân, Bác sĩ
Tiền điều kiện	Đã khám bệnh, đã có hồ sơ
Hậu điều kiện	Tin nhắn được gửi đến bác sĩ phụ trách
Luồng chính	<ol style="list-style-type: none"> Chọn mục “Tư vấn sau khám” Nhập nội dung Gửi đi Nhận phản hồi
Luồng thay thế	Bác sĩ không phản hồi trong thời gian quy định, chuyển sang tổng đài viên hỗ trợ

Use Case 9: Thanh toán hóa đơn

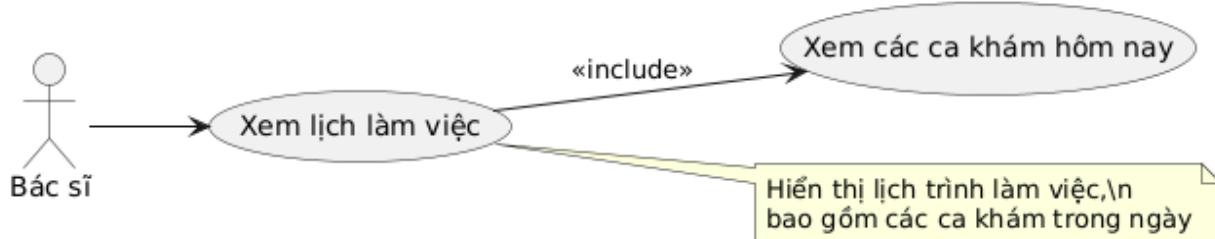


Mục	Nội dung
Tên Use Case	Thanh toán hóa đơn

Mục	Nội dung
Mô tả	Bệnh nhân thanh toán chi phí khám chữa bệnh qua online (VNPay, Momo, ngân hàng) hoặc tại cơ sở
Tác nhân liên quan	Bệnh nhân
Tiền điều kiện	Có hóa đơn phát sinh
Hậu điều kiện	Giao dịch thành công, hệ thống lưu trạng thái “Đã thanh toán”
Luồng chính	1. Truy cập mục “Hóa đơn” 2. Chọn phương thức thanh toán 3. Xác nhận giao dịch 4. Nhận thông báo thành công
Luồng thay thế	Giao dịch thất bại, yêu cầu thanh toán lại hoặc chọn phương thức khác

1.1.3.2 Use Case Diagram – Tác nhân: Bác sĩ (Doctor)

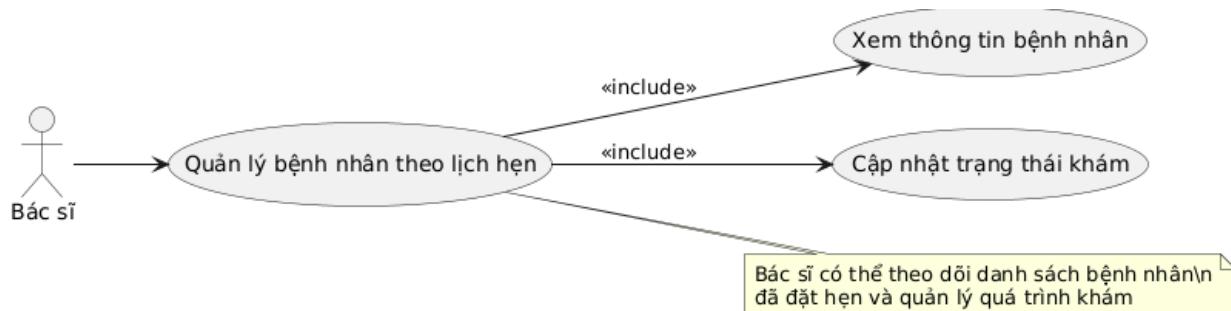
Use Case 1: Xem lịch làm việc và các ca khám hôm nay



Thuộc tính	Mô tả
Tên	Xem lịch làm việc và các ca khám hôm nay
Mã UC	UC_DOCTOR_01
Tác nhân chính	Bác sĩ
Mục tiêu	Cung cấp thông tin về lịch làm việc và danh sách bệnh nhân sẽ khám trong ngày.
Mô tả ngắn	Hệ thống cho phép bác sĩ xem lịch làm việc hàng ngày kèm theo danh sách các bệnh nhân đã đặt lịch khám.
Tiền điều kiện	Bác sĩ đã đăng nhập vào hệ thống.
Hậu điều kiện	Lịch làm việc và các ca khám trong ngày được hiển thị.

Thuộc tính	Mô tả
Luồng sự kiện chính	1. Bác sĩ đăng nhập vào hệ thống. 2. Bác sĩ chọn mục "Lịch làm việc". 3. Hệ thống hiển thị danh sách các ca khám trong ngày.
Luồng phụ / mở rộng	Nếu không có lịch hôm nay, hệ thống sẽ thông báo "Không có ca khám nào hôm nay".

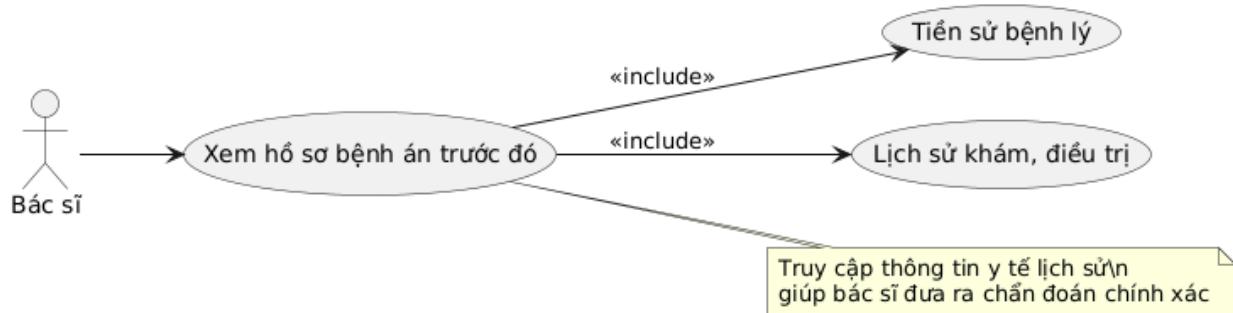
Use Case 2: Quản lý danh sách bệnh nhân theo lịch hẹn



Thuộc tính	Mô tả
Tên	Quản lý danh sách bệnh nhân theo lịch hẹn
Mã UC	UC_DOCTOR_02
Tác nhân chính	Bác sĩ
Mục tiêu	Quản lý bệnh nhân đến khám theo danh sách lịch hẹn đã lên.
Mô tả ngắn	Bác sĩ có thể xem, xác nhận, đánh dấu đã khám hoặc hủy lịch khám cho từng bệnh nhân.
Tiền điều kiện	Bác sĩ đã đăng nhập hệ thống và có lịch hẹn với bệnh nhân.
Hậu điều kiện	Danh sách bệnh nhân được cập nhật trạng thái đã khám, đang chờ khám hoặc hủy.
Luồng sự kiện chính	1. Bác sĩ vào chức năng "Quản lý bệnh nhân". 2. Bác sĩ xem danh sách bệnh nhân theo lịch hẹn. 3. Bác sĩ xác nhận hoặc hủy các lịch khám.

Thuộc tính	Mô tả
Luồng phụ / mở rộng	Nếu bệnh nhân không đến đúng giờ, bác sĩ có thể chọn hủy lịch hẹn và thông báo cho bệnh nhân.

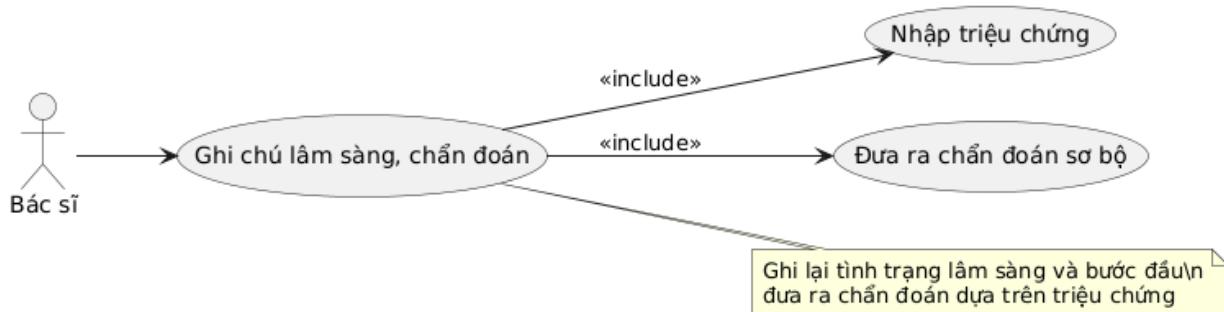
Use Case 3: Xem hồ sơ bệnh án trước đó của bệnh nhân



Thuộc tính	Mô tả
Tên	Xem hồ sơ bệnh án trước đó của bệnh nhân
Mã UC	UC_DOCTOR_03
Tác nhân chính	Bác sĩ
Mục tiêu	Xem hồ sơ bệnh án của bệnh nhân trước đó để đưa ra quyết định chẩn đoán chính xác.
Mô tả ngắn	Bác sĩ có thể truy cập và xem lại hồ sơ bệnh án cũ của bệnh nhân để tham khảo lịch sử bệnh lý.
Tiền điều kiện	Bác sĩ đã đăng nhập vào hệ thống và bệnh nhân đã có hồ sơ bệnh án trước đó.
Hậu điều kiện	Hồ sơ bệnh án trước đó được hiển thị đầy đủ cho bác sĩ.
Luồng sự kiện chính	1. Bác sĩ chọn bệnh nhân trong danh sách lịch hẹn. 2. Bác sĩ vào mục "Hồ sơ bệnh án". 3. Hệ thống hiển thị hồ sơ bệnh án trước đó của bệnh nhân.

Thuộc tính	Mô tả
Luồng phụ / mở rộng	Nếu bệnh nhân chưa có hồ sơ bệnh án, hệ thống sẽ thông báo rằng không có dữ liệu hồ sơ.

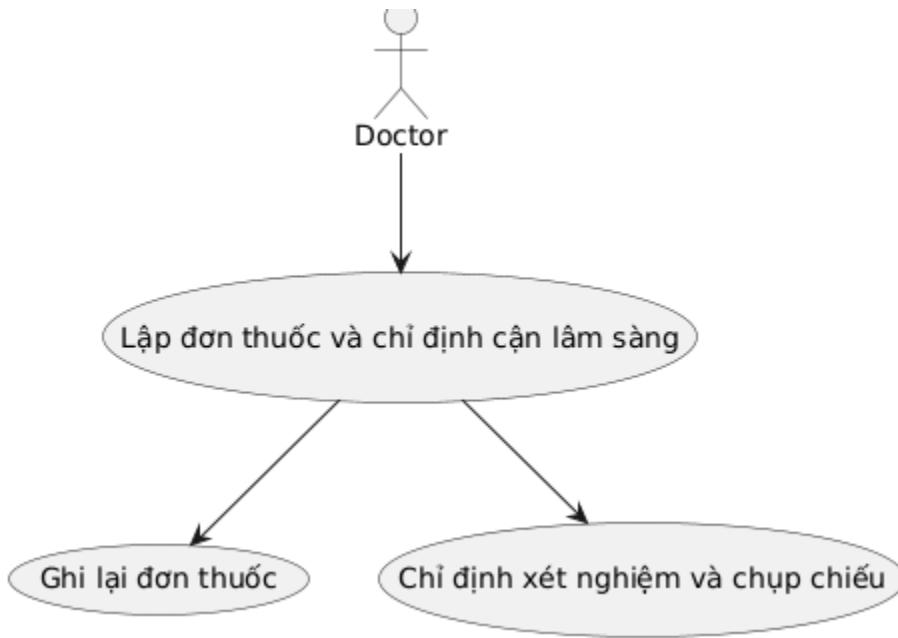
Use Case 4: Ghi chú lâm sàng, chẩn đoán bệnh



Thuộc tính	Mô tả
Tên	Ghi chú lâm sàng, chẩn đoán bệnh
Mã UC	UC_DOCTOR_04
Tác nhân chính	Bác sĩ
Mục tiêu	Ghi lại thông tin lâm sàng và đưa ra chẩn đoán bệnh của bệnh nhân.
Mô tả ngắn	Bác sĩ có thể ghi chú các triệu chứng, tình trạng của bệnh nhân và chẩn đoán bệnh sau khi kiểm tra.
Tiền điều kiện	Bác sĩ đã tiếp xúc và kiểm tra bệnh nhân, hệ thống đã có các thông tin về bệnh nhân.
Hậu điều kiện	Ghi chú lâm sàng và chẩn đoán bệnh của bệnh nhân được lưu lại và có thể tham khảo sau này.
Luồng sự kiện chính	1. Bác sĩ vào mục "Ghi chú lâm sàng". 2. Bác sĩ nhập các triệu chứng và chẩn đoán bệnh. 3. Hệ thống lưu lại các thông tin này vào hồ sơ bệnh án của bệnh nhân.

Thuộc tính	Mô tả
Luồng phụ / mở rộng	Bác sĩ có thể cập nhật hoặc chỉnh sửa ghi chú nếu có thêm thông tin sau các lần kiểm tra sau.

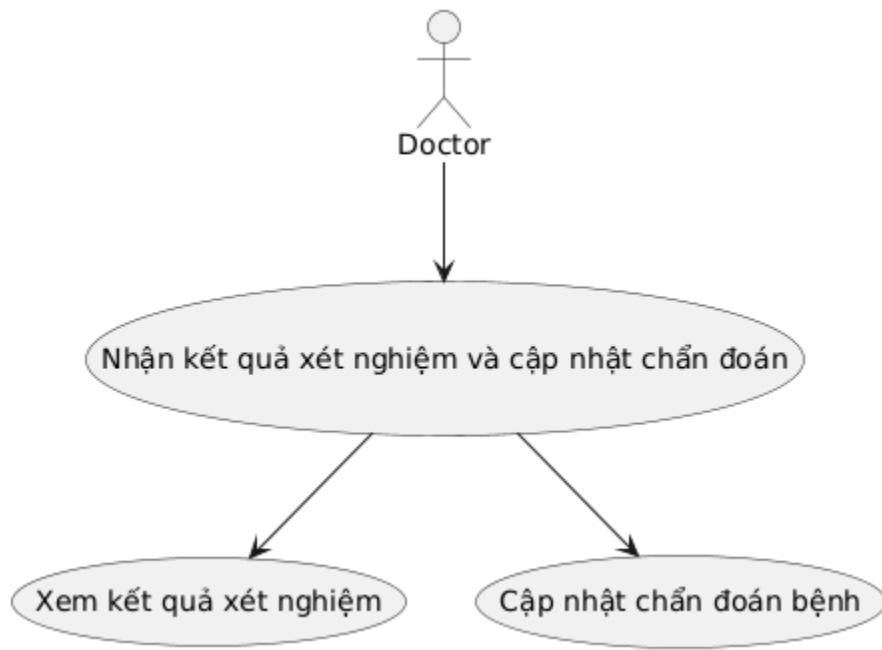
Use Case 5: Lập đơn thuốc và chỉ định cận lâm sàng (xét nghiệm, chụp chiếu)



Thuộc tính	Mô tả
Tên	Lập đơn thuốc và chỉ định cận lâm sàng (xét nghiệm, chụp chiếu)
Mã UC	UC_DOCTOR_05
Tác nhân chính	Bác sĩ
Mục tiêu	Bác sĩ lập đơn thuốc cho bệnh nhân và chỉ định các xét nghiệm, chụp chiếu cần thiết.
Mô tả ngắn	Bác sĩ sẽ căn cứ vào tình trạng bệnh lý của bệnh nhân để lập đơn thuốc và chỉ định xét nghiệm, chụp chiếu phù hợp.

Thuộc tính	Mô tả
Tiền điều kiện	Bác sĩ đã khám và đưa ra chẩn đoán cho bệnh nhân.
Hậu điều kiện	Đơn thuốc và chỉ định cận lâm sàng được ghi lại và có thể chuyển cho bệnh nhân.
Luồng sự kiện chính	1. Bác sĩ chọn mục "Lập đơn thuốc". 2. Bác sĩ chỉ định thuốc, xét nghiệm, và chụp chiếu. 3. Hệ thống lưu lại đơn thuốc và chỉ định cận lâm sàng.
Luồng phụ / mở rộng	Nếu bác sĩ cần chỉnh sửa đơn thuốc hoặc chỉ định cận lâm sàng, hệ thống cho phép chỉnh sửa.

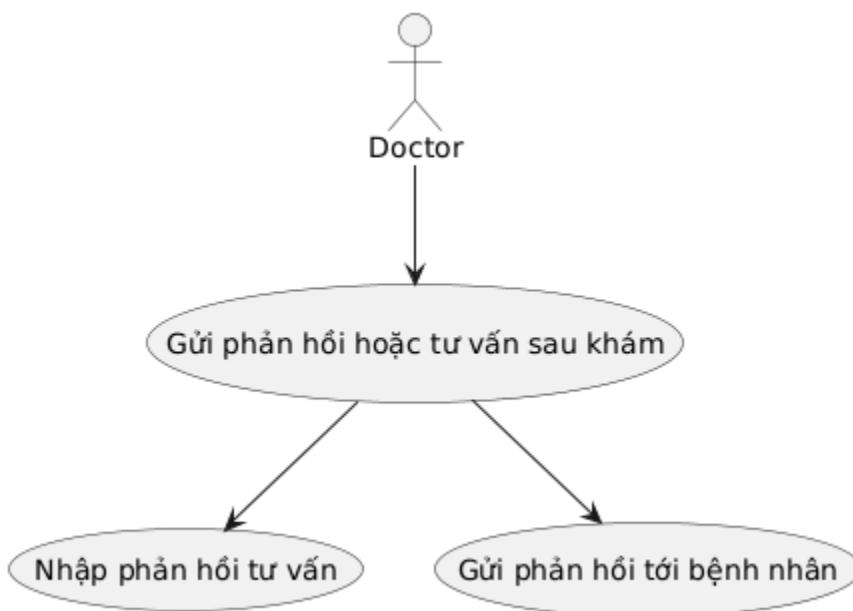
Use Case 6: Nhận kết quả xét nghiệm và cập nhật chẩn đoán



Thuộc tính	Mô tả
Tên	Nhận kết quả xét nghiệm và cập nhật chẩn đoán
Mã UC	UC_DOCTOR_06

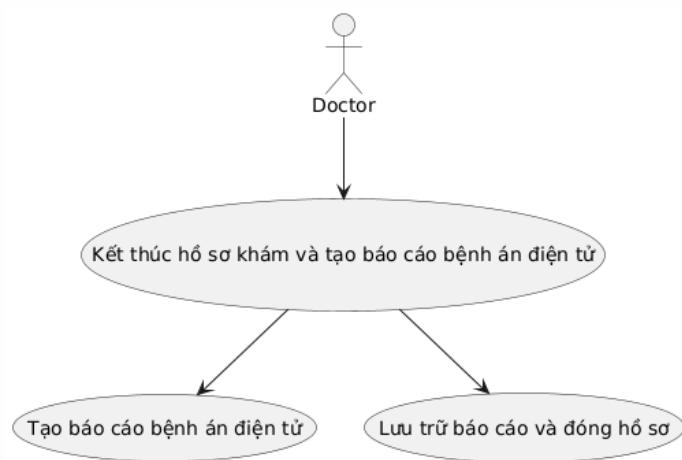
Thuộc tính	Mô tả
Tác nhân chính	Bác sĩ
Mục tiêu	Bác sĩ nhận kết quả xét nghiệm của bệnh nhân và cập nhật lại chẩn đoán nếu cần.
Mô tả ngắn	Bác sĩ có thể nhận kết quả xét nghiệm từ hệ thống và căn cứ vào đó để thay đổi hoặc bổ sung chẩn đoán bệnh lý.
Tiền điều kiện	Bệnh nhân đã thực hiện xét nghiệm và hệ thống đã cập nhật kết quả.
Hậu điều kiện	Bác sĩ cập nhật lại chẩn đoán và thông tin bệnh án.
Luồng sự kiện chính	1. Bác sĩ vào mục "Kết quả xét nghiệm". 2. Bác sĩ nhận kết quả xét nghiệm từ hệ thống. 3. Bác sĩ cập nhật lại chẩn đoán bệnh dựa trên kết quả.
Luồng phụ / mở rộng	Bác sĩ có thể yêu cầu xét nghiệm bổ sung nếu kết quả không đủ rõ ràng.

Use Case 7: Gửi phản hồi hoặc tư vấn sau khám cho bệnh nhân



Thuộc tính	Mô tả
Tên	Gửi phản hồi hoặc tư vấn sau khám cho bệnh nhân
Mã UC	UC_DOCTOR_07
Tác nhân chính	Bác sĩ
Mục tiêu	Bác sĩ gửi phản hồi về tình trạng sức khỏe hoặc tư vấn tiếp theo cho bệnh nhân.
Mô tả ngắn	Bác sĩ gửi phản hồi hoặc hướng dẫn tiếp theo cho bệnh nhân sau khi khám xong, giúp bệnh nhân hiểu rõ tình trạng và kế hoạch điều trị.
Tiền điều kiện	Bác sĩ đã khám và có kết luận về tình trạng sức khỏe của bệnh nhân.
Hậu điều kiện	Phản hồi hoặc tư vấn được gửi tới bệnh nhân qua hệ thống.
Luồng sự kiện chính	1. Bác sĩ vào mục "Phản hồi sau khám". 2. Bác sĩ nhập thông tin tư vấn hoặc phản hồi. 3. Hệ thống gửi phản hồi cho bệnh nhân.
Luồng phụ / mở rộng	Nếu bác sĩ cần giải thích thêm, hệ thống cho phép gửi phản hồi qua email hoặc tin nhắn.

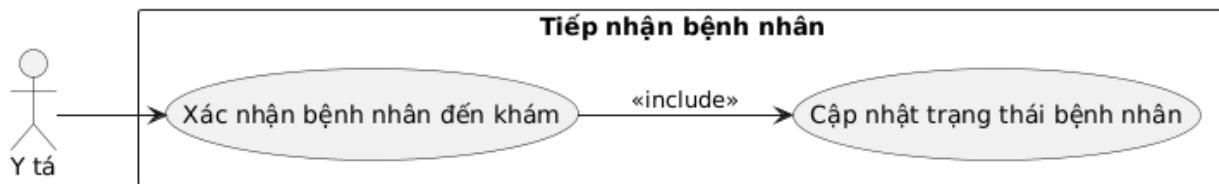
Use Case 8: Kết thúc hồ sơ khám và tạo báo cáo bệnh án điện tử



Thuộc tính	Mô tả
Tên	Kết thúc hồ sơ khám và tạo báo cáo bệnh án điện tử
Mã UC	UC_DOCTOR_08
Tác nhân chính	Bác sĩ
Mục tiêu	Bác sĩ kết thúc hồ sơ khám và tạo báo cáo bệnh án điện tử cho bệnh nhân.
Mô tả ngắn	Sau khi hoàn thành khám và các thủ tục liên quan, bác sĩ sẽ tạo báo cáo bệnh án điện tử cho bệnh nhân và đóng hồ sơ.
Tiền điều kiện	Bác sĩ đã hoàn thành việc khám và cập nhật các thông tin cần thiết.
Hậu điều kiện	Hồ sơ khám được đóng và báo cáo bệnh án được lưu trữ trong hệ thống.
Luồng sự kiện chính	1. Bác sĩ vào mục "Kết thúc hồ sơ khám". 2. Bác sĩ tạo báo cáo bệnh án điện tử. 3. Hệ thống lưu trữ báo cáo và đóng hồ sơ khám.
Luồng phụ / mở rộng	Hệ thống có thể yêu cầu bác sĩ xác nhận lại thông tin trước khi đóng hồ sơ.

1.1.3.3 Use Case Diagram – Tác nhân: Y tá (nurse)

Use Case 1: Tiếp nhận bệnh nhân tại phòng khám hoặc từ hệ thống



Thuộc tính	Mô tả
Tên	Tiếp nhận bệnh nhân
Mã UC	UC_NURSE_01

Thuộc tính	Mô tả
Tác nhân chính	Y tá
Mục tiêu	Tiếp nhận bệnh nhân và cập nhật trạng thái vào hệ thống để bắt đầu quy trình khám chữa bệnh
Mô tả ngắn	Y tá xác nhận sự có mặt của bệnh nhân tại cơ sở y tế và cập nhật trạng thái vào hệ thống.
Tiền điều kiện	Bệnh nhân đã đặt lịch hoặc đến trực tiếp
Hậu điều kiện	Bệnh nhân được gắn vào quy trình khám bệnh, thông báo bác sĩ sẵn sàng khám
Luồng chính	<ul style="list-style-type: none"> 1. Y tá tìm lịch hẹn 2. Xác nhận thông tin bệnh nhân 3. Cập nhật trạng thái "Đã đến" hoặc "Chờ khám"
Luồng phụ	Nếu bệnh nhân chưa có lịch, y tá có thể thêm vào danh sách chờ khám trực tiếp

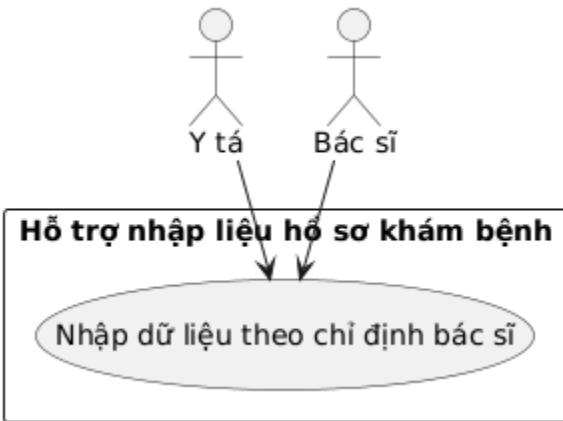
Use Case 2: Đo chỉ số sinh tồn và cập nhật hồ sơ



Thuộc tính	Mô tả
Tên	Đo chỉ số sinh tồn
Mã UC	UC_NURSE_02
Tác nhân chính	Y tá

Thuộc tính	Mô tả
Mục tiêu	Đo các chỉ số sức khỏe ban đầu và cập nhật vào hồ sơ bệnh nhân
Mô tả ngắn	Các chỉ số như huyết áp, nhiệt độ, mạch, SPO2 được đo và nhập vào hệ thống trước khi bác sĩ khám
Tiền điều kiện	Bệnh nhân đã tiếp nhận thành công
Hậu điều kiện	Hồ sơ bệnh án có đầy đủ chỉ số sinh tồn
Luồng chính	1. Y tá đo các chỉ số 2. Nhập vào hệ thống 3. Lưu vào hồ sơ khám của bệnh nhân
Luồng phụ	Nếu phát hiện bất thường, chuyển sang Use Case “Thông báo bác sĩ khi có bất thường”

Use Case 3: Hỗ trợ bác sĩ nhập dữ liệu hồ sơ khám bệnh



Thuộc tính	Mô tả
Tên	Hỗ trợ nhập dữ liệu hồ sơ khám
Mã UC	UC_NURSE_03

Thuộc tính	Mô tả
Tác nhân chính	Y tá
Mục tiêu	Hỗ trợ bác sĩ trong quá trình nhập liệu, ghi chép kết quả khám bệnh
Mô tả ngắn	Y tá phối hợp với bác sĩ để ghi nhận kết quả chẩn đoán, đơn thuốc, chỉ định xét nghiệm...
Tiền điều kiện	Bác sĩ đang trong quá trình khám bệnh
Hậu điều kiện	Dữ liệu khám bệnh được cập nhật đầy đủ và chính xác
Luồng chính	<ul style="list-style-type: none"> 1. Y tá mở hồ sơ bệnh án 2. Ghi lại kết luận của bác sĩ theo lời đọc 3. Cập nhật vào hệ thống
Luồng phụ	Y tá có thể lưu nháp nếu bác sĩ cần chỉnh sửa sau

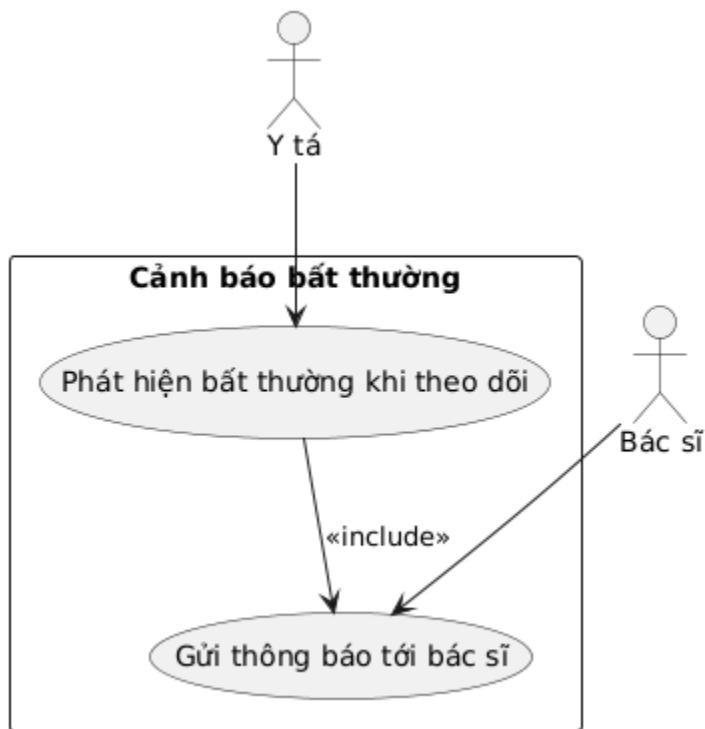
Use Case 4: Theo dõi bệnh nhân nội trú



Thuộc tính	Mô tả
Tên	Theo dõi bệnh nhân nội trú
Mã UC	UC_NURSE_04
Tác nhân chính	Y tá
Mục tiêu	Ghi nhận tình trạng của bệnh nhân đang nằm viện
Mô tả ngắn	Y tá theo dõi diễn biến lâm sàng, nhập chỉ số theo dõi hằng ngày, hỗ trợ chăm sóc bệnh nhân nội trú
Tiền điều kiện	Bệnh nhân đang trong danh sách điều trị nội trú
Hậu điều kiện	Hồ sơ theo dõi được cập nhật đầy đủ mỗi ngày

Thuộc tính	Mô tả
Luồng chính	1. Y tá truy cập hồ sơ bệnh nhân nội trú 2. Ghi nhận các chỉ số, biểu hiện 3. Cập nhật thông tin chăm sóc, thuốc sử dụng...
Luồng phụ	Gửi cảnh báo nếu có dấu hiệu xấu, chuyển sang Use Case "Thông báo cho bác sĩ khi có bất thường"

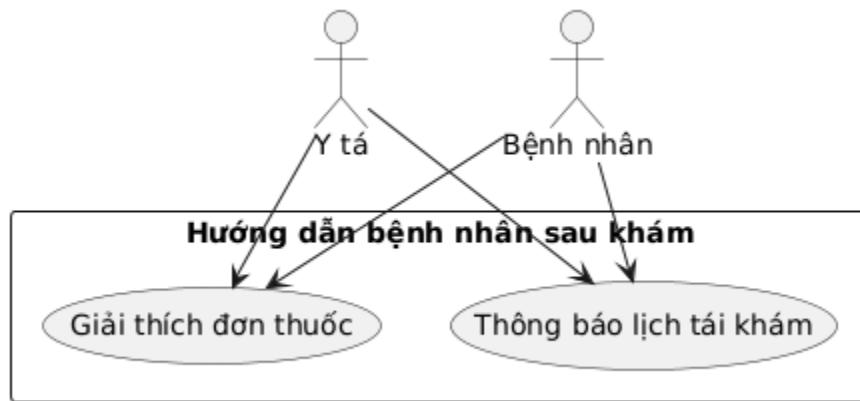
Use Case 5: Thông báo cho bác sĩ khi có bất thường



Thuộc tính	Mô tả
Tên	Thông báo khi có bất thường
Mã UC	UC_NURSE_05
Tác nhân chính	Y tá
Mục tiêu	Gửi cảnh báo cho bác sĩ nếu tình trạng bệnh nhân có vấn đề nghiêm trọng

Thuộc tính	Mô tả
Mô tả ngắn	Y tá là người theo dõi đầu tiên, khi phát hiện bất thường về sinh hiệu hoặc triệu chứng, cần nhanh chóng thông báo cho bác sĩ
Tiền điều kiện	Bệnh nhân đang được theo dõi, và dữ liệu đã được đo
Hậu điều kiện	Bác sĩ nhận được cảnh báo và có thể can thiệp kịp thời
Luồng chính	<ol style="list-style-type: none"> 1. Y tá phát hiện bất thường 2. Gửi thông báo qua hệ thống hoặc gọi trực tiếp bác sĩ 3. Cập nhật thông tin cảnh báo vào hồ sơ
Luồng phụ	Nếu bác sĩ không phản hồi, hệ thống có thể gửi nhắc lại tự động

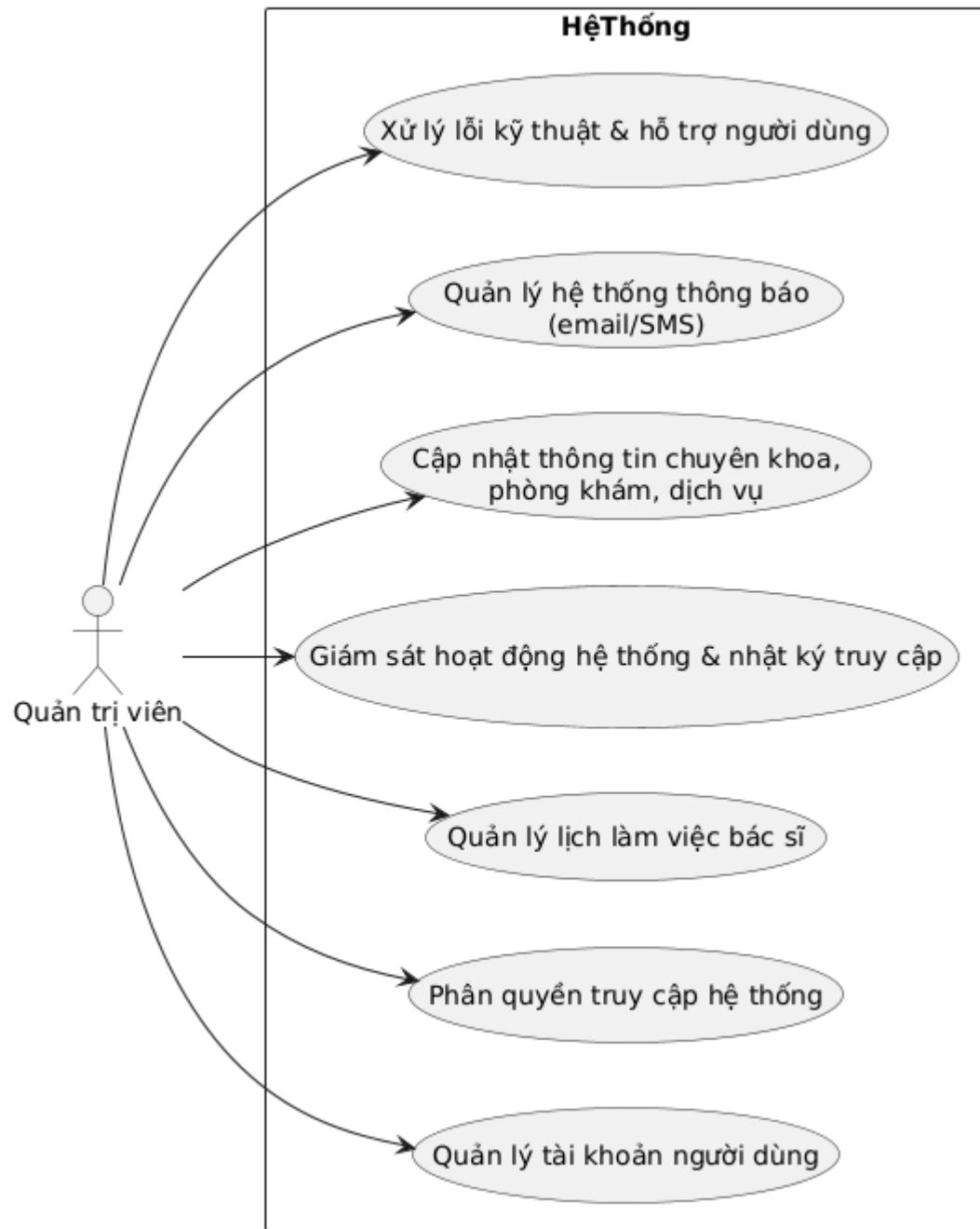
Use Case 6: Hướng dẫn bệnh nhân về đơn thuốc và lịch tái khám



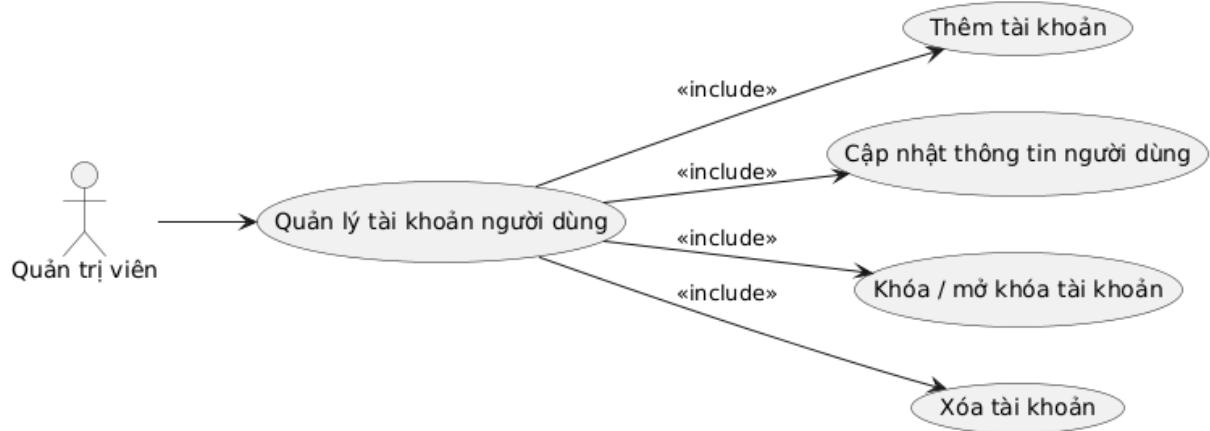
Thuộc tính	Mô tả
Tên	Hướng dẫn đơn thuốc và tái khám
Mã UC	UC_NURSE_06
Tác nhân chính	Y tá

Thuộc tính	Mô tả
Mục tiêu	Giải thích rõ cách dùng thuốc và hẹn lịch tái khám cho bệnh nhân sau khi rời phòng khám
Mô tả ngắn	Y tá giúp bệnh nhân hiểu rõ về đơn thuốc (liều lượng, thời gian uống) và tư vấn lịch tái khám theo chỉ định của bác sĩ
Tiền điều kiện	Đã có đơn thuốc và chỉ định tái khám từ bác sĩ
Hậu điều kiện	Bệnh nhân hiểu rõ cách điều trị và tái khám đúng hạn
Luồng chính	<ol style="list-style-type: none"> 1. Y tá lấy đơn thuốc từ hệ thống 2. Hướng dẫn bệnh nhân chi tiết 3. Thông báo lịch tái khám nếu có
Luồng phụ	Y tá in đơn thuốc nếu bệnh nhân yêu cầu bản giấy

1.1.3.4 Use Case Diagram – Tác nhân: Adminitistrator

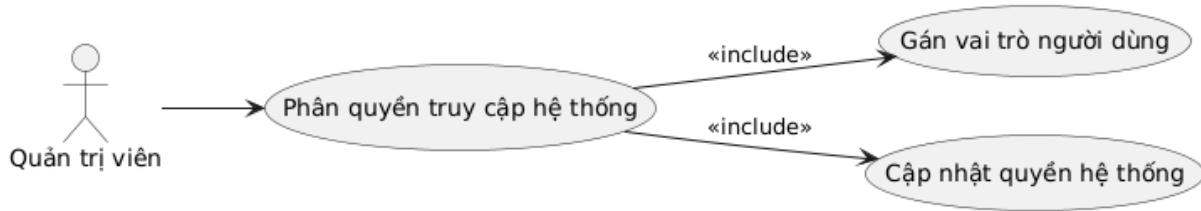


Use Case 1: Quản lý tài khoản người dùng



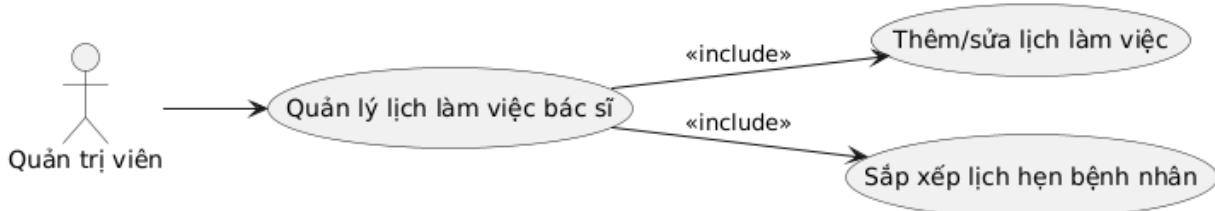
Thuộc tính	Nội dung
Tên Use Case	Quản lý tài khoản người dùng
Mô tả	Quản trị viên có thể thêm mới, cập nhật, khóa/mở khóa hoặc xóa tài khoản cho các vai trò khác nhau như bệnh nhân, bác sĩ, y tá, dược sĩ,...
Tác nhân chính	Quản trị viên
Tiền điều kiện	Quản trị viên đã đăng nhập thành công vào hệ thống
Luồng sự kiện chính	1. Truy cập chức năng quản lý người dùng 2. Chọn vai trò người dùng 3. Thực hiện thao tác (thêm, sửa, khóa, xóa) 4. Xác nhận lưu thông tin
Luồng phụ (ngoại lệ)	- Tài khoản đã tồn tại - Không có quyền thao tác trên tài khoản hệ thống
Kết quả mong đợi	Tài khoản người dùng được cập nhật đúng theo yêu cầu

Use Case 2: Phân quyền truy cập hệ thống



Thuộc tính	Nội dung
Tên Use Case	Phân quyền truy cập hệ thống
Mô tả	Cho phép quản trị viên thiết lập quyền truy cập cho từng tài khoản hoặc nhóm tài khoản
Tác nhân chính	Quản trị viên
Tiền điều kiện	Có tài khoản quản trị với quyền cao nhất
Luồng sự kiện chính	<ol style="list-style-type: none"> Chọn người dùng hoặc nhóm quyền Gán quyền cho chức năng tương ứng Lưu thay đổi
Luồng phụ (ngoại lệ)	<ul style="list-style-type: none"> Gán quyền không hợp lệ Tài khoản không tồn tại
Kết quả mong đợi	Tài khoản người dùng có đúng quyền và chức năng được phép sử dụng

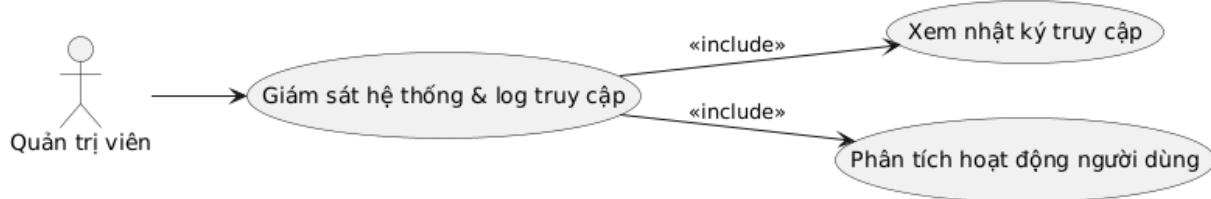
Use Case 3: Quản lý lịch làm việc của bác sĩ, sắp xếp lịch hẹn



Thuộc tính	Nội dung
Tên Use Case	Quản lý lịch làm việc của bác sĩ, sắp xếp lịch hẹn

Thuộc tính	Nội dung
Mô tả	Cập nhật lịch làm việc định kỳ hoặc đặc biệt của bác sĩ. Hệ thống tự động kiểm tra xung đột lịch và hỗ trợ sắp xếp lịch hẹn khám cho bệnh nhân
Tác nhân chính	Quản trị viên
Tiền điều kiện	Bác sĩ đã có tài khoản trong hệ thống và có chuyên khoa xác định
Luồng sự kiện chính	<ol style="list-style-type: none"> 1. Chọn bác sĩ cần cập nhật lịch 2. Thiết lập lịch làm việc theo ca/ngày 3. Hệ thống kiểm tra xung đột 4. Sắp xếp lịch hẹn tự động
Luồng phụ (ngoại lệ)	<ul style="list-style-type: none"> - Ca khám trùng giờ - Không có phòng khám trống
Kết quả mong đợi	Lịch làm việc của bác sĩ được cập nhật và lịch hẹn được sắp xếp khoa học

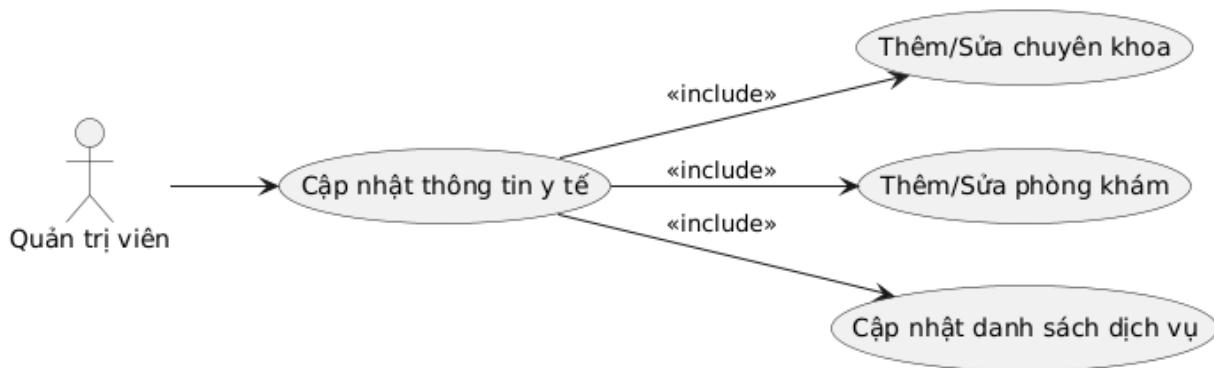
Use Case 4: Giám sát hoạt động hệ thống và nhật ký truy cập (log)



Thuộc tính	Nội dung
Tên Use Case	Giám sát hoạt động hệ thống và nhật ký truy cập
Mô tả	Ghi lại tất cả hành động quan trọng của người dùng và cung cấp báo cáo truy cập cho quản trị viên nhằm phát hiện hành vi bất thường hoặc lỗi hệ thống
Tác nhân chính	Quản trị viên

Thuộc tính	Nội dung
Tiền điều kiện	Đăng nhập hệ thống với quyền quản trị
Luồng sự kiện chính	1. Mở bảng điều khiển giám sát hệ thống 2. Xem log truy cập theo bộ lọc 3. Phân tích các hành vi bất thường
Luồng phụ (ngoại lệ)	- Nhật ký bị lỗi hoặc không ghi nhận - Quá tải dữ liệu log
Kết quả mong đợi	Quản trị viên kiểm soát được hoạt động và truy vết được lỗi nếu xảy ra

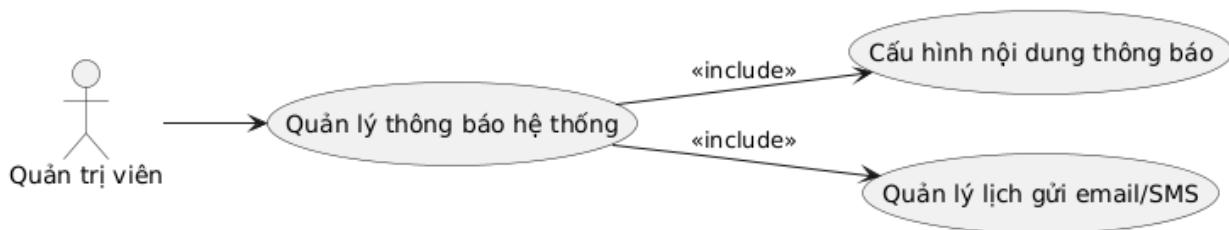
Use Case 5: Cập nhật thông tin chuyên khoa, phòng khám, dịch vụ



Thuộc tính	Nội dung
Tên Use Case	Cập nhật thông tin chuyên khoa, phòng khám, dịch vụ
Mô tả	Quản trị viên có thể thêm mới, chỉnh sửa hoặc xóa thông tin về chuyên khoa, phòng khám, dịch vụ y tế đang được cung cấp trong hệ thống
Tác nhân chính	Quản trị viên
Tiền điều kiện	Quản trị viên đã đăng nhập hệ thống

Thuộc tính	Nội dung
Luồng sự kiện chính	1. Truy cập vào giao diện quản lý danh mục 2. Chọn chuyên mục cần quản lý 3. Thêm/sửa/xóa thông tin 4. Lưu lại thay đổi
Luồng phụ (ngoại lệ)	- Thông tin nhập vào không hợp lệ - Không thể xóa chuyên khoa đang có bác sĩ hoạt động
Kết quả mong đợi	Thông tin được cập nhật đúng và hiển thị chính xác trong các chức năng liên quan

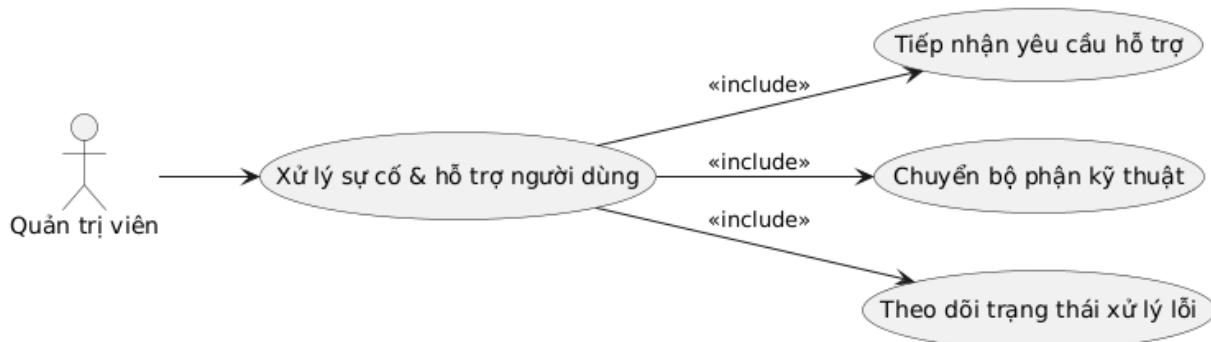
Use Case 6: Quản lý hệ thống email/sms thông báo



Thuộc tính	Nội dung
Tên Use Case	Quản lý hệ thống email/sms thông báo
Mô tả	Quản trị viên cấu hình hệ thống gửi email và SMS thông báo như lịch hẹn, đơn thuốc, kết quả khám, phản hồi...
Tác nhân chính	Quản trị viên
Tiền điều kiện	Hệ thống đã tích hợp API email/SMS hoặc các cổng gửi thông báo
Luồng sự kiện chính	1. Mở phần cấu hình thông báo 2. Chọn hình thức gửi (email/SMS) 3. Thiết lập nội dung mẫu, thời điểm gửi 4. Lưu và kiểm tra gửi thử

Thuộc tính	Nội dung
Luồng phụ (ngoại lệ)	- Lỗi kết nối với hệ thống gửi - Mẫu nội dung sai định dạng hoặc trùng lặp
Kết quả mong đợi	Hệ thống gửi thông báo tự động đến người dùng theo đúng cấu hình

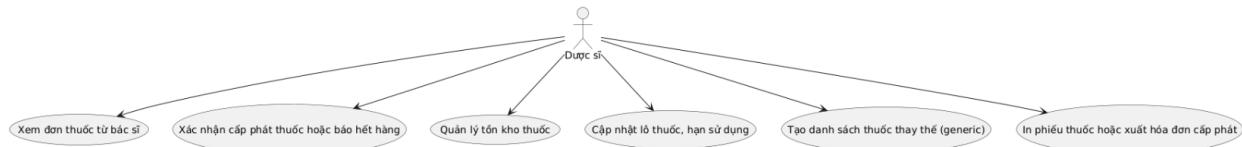
Use Case 7: Xử lý các lỗi kỹ thuật hoặc yêu cầu hỗ trợ từ người dùng



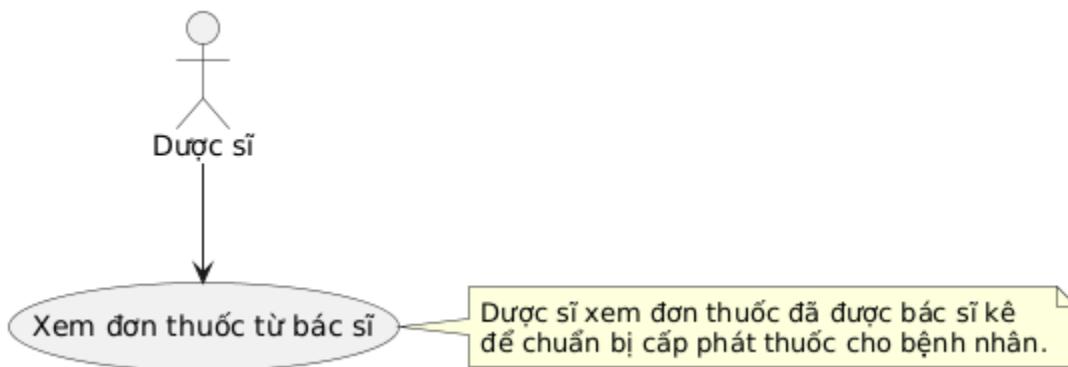
Thuộc tính	Nội dung
Tên Use Case	Xử lý các lỗi kỹ thuật hoặc yêu cầu hỗ trợ từ người dùng
Mô tả	Quản trị viên tiếp nhận và xử lý các yêu cầu kỹ thuật do người dùng gửi lên, có thể tự xử lý hoặc chuyển cho bộ phận kỹ thuật
Tác nhân chính	Quản trị viên
Tiền điều kiện	Người dùng đã gửi phản hồi thông qua hệ thống hỗ trợ
Luồng sự kiện chính	<ol style="list-style-type: none"> Truy cập hệ thống hỗ trợ kỹ thuật Xem danh sách lỗi/yêu cầu Phân loại và xử lý hoặc chuyển bộ phận liên quan Cập nhật trạng thái yêu cầu
Luồng phụ (ngoại lệ)	<ul style="list-style-type: none"> Yêu cầu không đủ thông tin Lỗi kỹ thuật chưa có hướng xử lý

Thuộc tính	Nội dung
Kết quả mong đợi	Yêu cầu hỗ trợ được xử lý hoặc phản hồi cho người dùng đúng thời gian và nội dung phù hợp

1.1.3.4 Use Case Diagram – Tác nhân: Dược sĩ



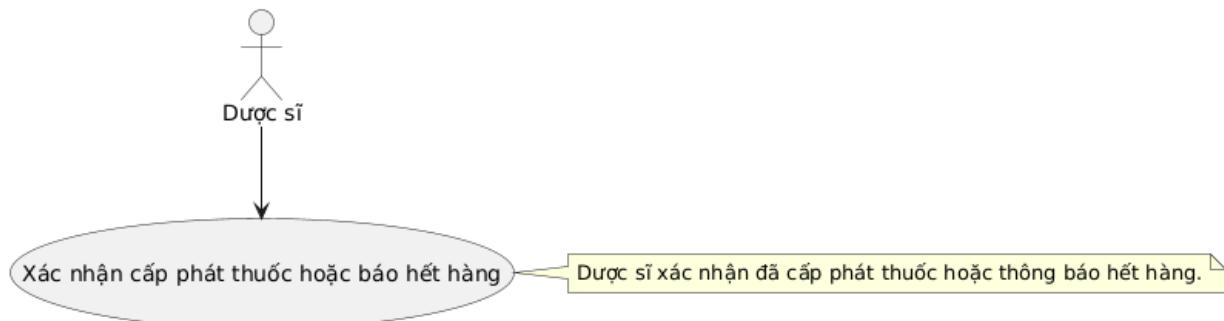
Use Case 1: Xem đơn thuốc từ bác sĩ



Mục	Nội dung
Tên Use Case	Xem đơn thuốc từ bác sĩ
Actor chính	Dược sĩ
Mô tả	Dược sĩ xem đơn thuốc đã được bác sĩ kê sau khi khám bệnh, nhằm chuẩn bị cấp phát thuốc cho bệnh nhân.
Luồng sự kiện chính	<ol style="list-style-type: none"> Dược sĩ đăng nhập vào hệ thống. Chọn mục "Đơn thuốc". Xem danh sách đơn thuốc mới. Chọn đơn cụ thể để xem chi tiết.

Mục	Nội dung
Luồng phụ / mở rộng	- Xem đơn thuốc theo ngày, mã bệnh nhân, tên bác sĩ.
Điều kiện tiên quyết	Dược sĩ đã đăng nhập. Có đơn thuốc do bác sĩ gửi.
Kết quả	Hiển thị thông tin đơn thuốc chi tiết.

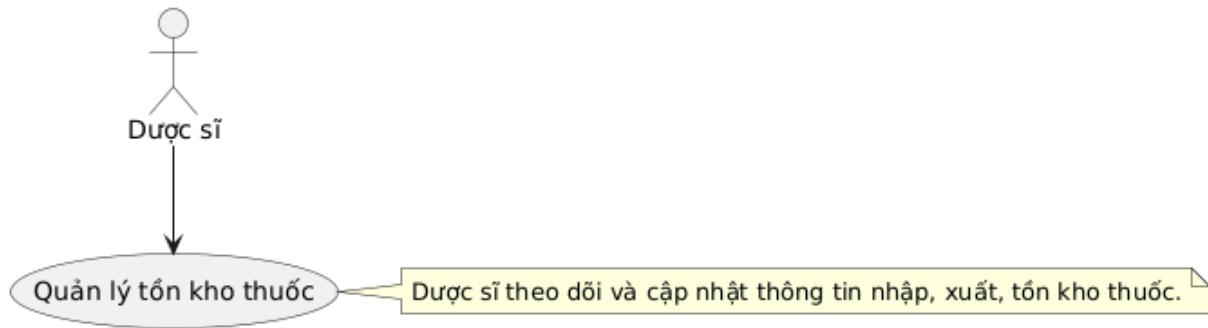
Use Case 2: Xác nhận cấp phát thuốc hoặc báo hết hàng



Mục	Nội dung
Tên Use Case	Xác nhận cấp phát thuốc hoặc báo hết hàng
Actor chính	Dược sĩ
Mô tả	Sau khi xem đơn thuốc, dược sĩ xác nhận đã cấp phát hoặc thông báo thuốc không đủ.
Luồng sự kiện chính	<ol style="list-style-type: none"> Dược sĩ chọn đơn thuốc. Kiểm tra kho thuốc. Xác nhận đã cấp phát hoặc báo hết hàng. Gửi thông báo đến bác sĩ/bệnh nhân.
Luồng phụ / mở rộng	- Tự động gửi đề xuất mua thêm nếu thuốc hết.
Điều kiện tiên quyết	Đã có đơn thuốc hợp lệ.

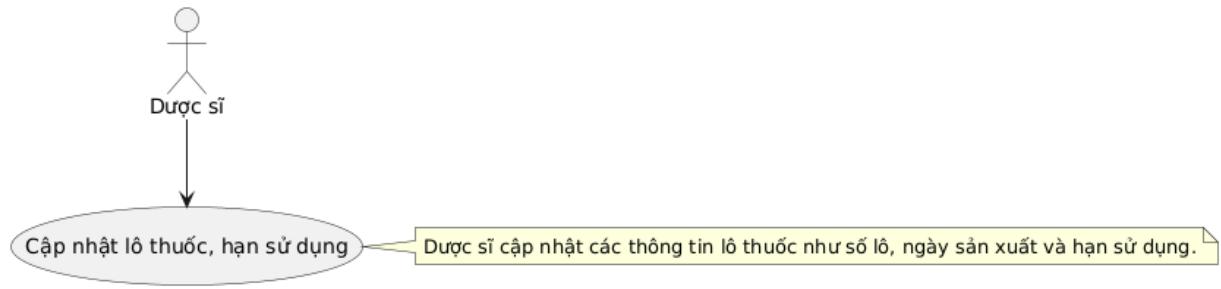
Mục	Nội dung
Kết quả	Cập nhật trạng thái đơn thuốc: đã cấp phát / không cấp phát.

Use Case 3: Quản lý tồn kho thuốc



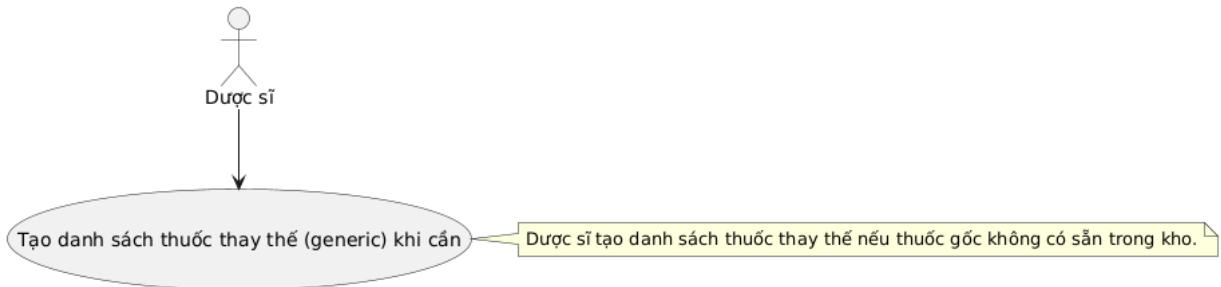
Mục	Nội dung
Tên Use Case	Quản lý tồn kho thuốc
Actor chính	Dược sĩ
Mô tả	Dược sĩ theo dõi và cập nhật tồn kho thuốc, bao gồm số lượng nhập, xuất và còn lại.
Luồng sự kiện chính	<ol style="list-style-type: none"> Vào mục quản lý kho. Nhập thông tin nhập/xuất thuốc. Hệ thống tính toán tồn kho.
Luồng phụ / mở rộng	- Xuất báo cáo tồn kho định kỳ.
Điều kiện tiên quyết	Có quyền truy cập kho thuốc.
Kết quả	Tồn kho thuốc được cập nhật đúng.

Use Case 4: Cập nhật lô thuốc, hạn sử dụng



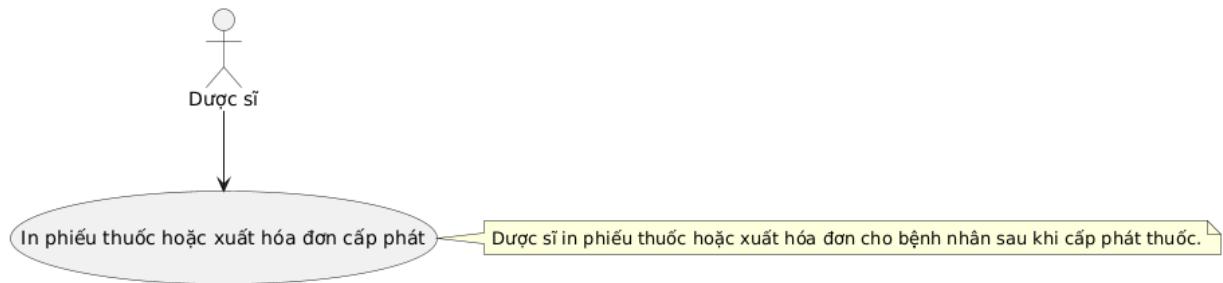
Mục	Nội dung
Tên Use Case	Cập nhật lô thuốc, hạn sử dụng
Actor chính	Dược sĩ
Mô tả	Ghi nhận các thông tin liên quan đến lô thuốc như số lô, ngày sản xuất, hạn dùng để quản lý chất lượng.
Luồng sự kiện chính	1. Dược sĩ thêm mới hoặc chỉnh sửa thông tin lô thuốc. 2. Nhập hạn sử dụng và các thông tin khác.
Luồng phụ / mở rộng	- Hệ thống cảnh báo nếu thuốc sắp hết hạn.
Điều kiện tiên quyết	Thuốc đã được nhập kho.
Kết quả	Cập nhật thông tin chi tiết cho từng lô thuốc.

Use Case 5: Tạo danh sách thuốc thay thế (generic)



Mục	Nội dung
Tên Use Case	Tạo danh sách thuốc thay thế (generic) khi cần
Actor chính	Dược sĩ
Mô tả	Khi thuốc gốc không có sẵn, dược sĩ có thể đề xuất thuốc tương đương về hoạt chất và hiệu quả.
Luồng sự kiện chính	<ol style="list-style-type: none"> 1. Dược sĩ chọn thuốc cần thay thế. 2. Chọn thuốc generic tương đương. 3. Cập nhật đơn thuốc và thông báo đến bác sĩ.
Luồng phụ / mở rộng	- Phê duyệt từ bác sĩ nếu thuốc thay thế khác biệt.
Điều kiện tiên quyết	Không có thuốc gốc trong kho.
Kết quả	Bệnh nhân được cấp thuốc thay thế phù hợp.

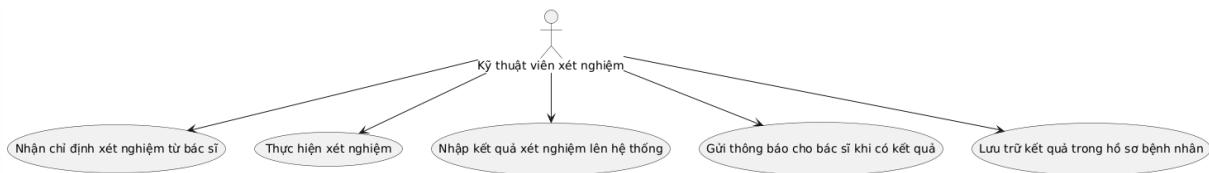
Use Case 6: In phiếu thuốc hoặc xuất hóa đơn cấp phát



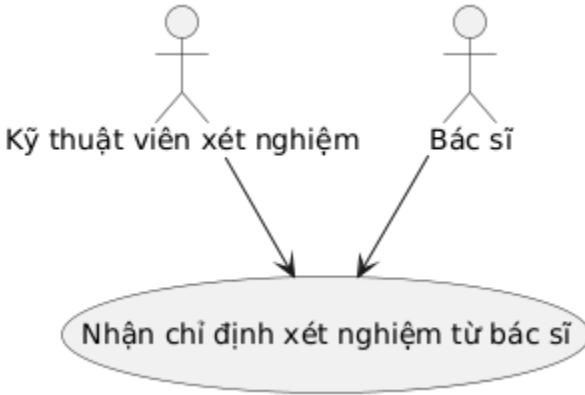
Mục	Nội dung
Tên Use Case	In phiếu thuốc hoặc xuất hóa đơn cấp phát
Actor chính	Dược sĩ
Mô tả	Sau khi xác nhận cấp phát thuốc, dược sĩ in phiếu cấp thuốc và hóa đơn tương ứng cho bệnh nhân.

Mục	Nội dung
Luồng sự kiện chính	1. Xác nhận cấp phát hoàn tất. 2. Chọn in phiếu hoặc xuất hóa đơn. 3. Hệ thống tạo bản in/pdf.
Luồng phụ / mở rộng	- Gửi hóa đơn qua email nếu cần.
Điều kiện tiên quyết	Đơn thuốc đã được cấp phát.
Kết quả	Bệnh nhân nhận được phiếu thuốc hoặc hóa đơn.

1.1.3.4 Use Case Diagram – Tác nhân: Kỹ thuật viên xét nghiệm



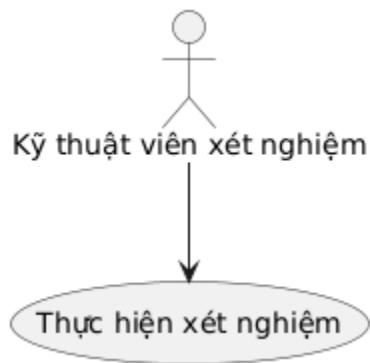
1. Use Case: Nhận chỉ định xét nghiệm từ bác sĩ



Tên Use Case	Nhận chỉ định xét nghiệm từ bác sĩ
Mô tả	Kỹ thuật viên nhận chỉ định xét nghiệm từ bác sĩ, bao gồm các xét nghiệm cụ thể (máu, nước tiểu, siêu âm, v.v.).

Tên Use Case	Nhận chỉ định xét nghiệm từ bác sĩ
Diễn viên	Kỹ thuật viên xét nghiệm, Bác sĩ
Luồng sự kiện chính	<ol style="list-style-type: none"> Bác sĩ xác định các xét nghiệm cần thiết cho bệnh nhân. Kỹ thuật viên nhận chỉ định từ bác sĩ. Kỹ thuật viên chuẩn bị các vật tư và thiết bị cần thiết cho xét nghiệm.
Điều kiện tiên quyết	Bác sĩ đã chuẩn đoán bệnh cho bệnh nhân và quyết định cần làm các xét nghiệm nào.
Điều kiện hậu quả	Kỹ thuật viên chuẩn bị sẵn sàng thực hiện xét nghiệm dựa trên chỉ định của bác sĩ.

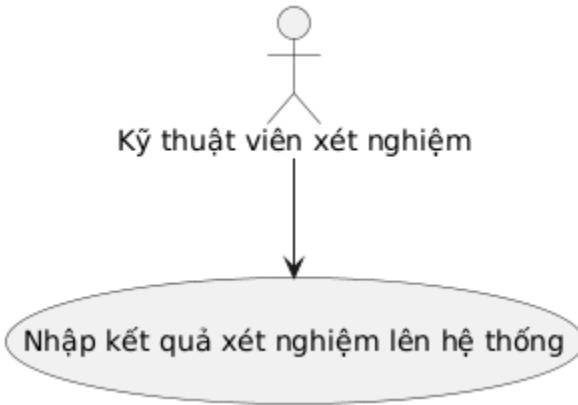
2. Use Case: Thực hiện xét nghiệm (máu, nước tiểu, chụp X-quang, siêu âm,...)



Tên Use Case	Thực hiện xét nghiệm (máu, nước tiểu, chụp X-quang, siêu âm,...)
Mô tả	Kỹ thuật viên thực hiện các loại xét nghiệm theo chỉ định của bác sĩ như xét nghiệm máu, nước tiểu, siêu âm, chụp X-quang, v.v.
Diễn viên	Kỹ thuật viên xét nghiệm
Luồng sự kiện chính	<ol style="list-style-type: none"> Kỹ thuật viên chuẩn bị cho xét nghiệm. Thực hiện các bước xét nghiệm như lấy mẫu, phân tích, chụp ảnh, v.v. Đảm bảo các thông số và yêu cầu kỹ thuật của xét nghiệm được đáp ứng đầy đủ.
Điều kiện tiên quyết	Chỉ định xét nghiệm đã được bác sĩ cung cấp và bệnh nhân đã có mặt để thực hiện xét nghiệm.

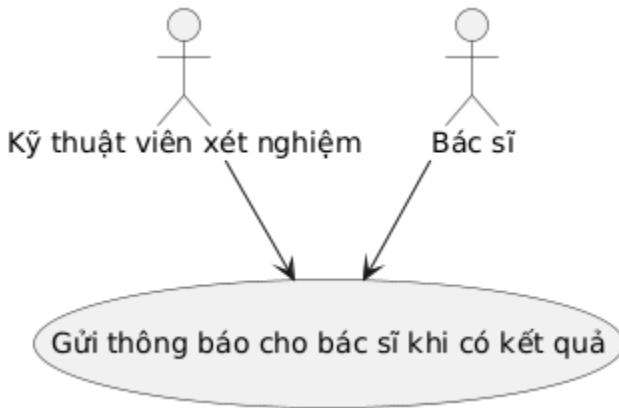
Tên Use Case	Thực hiện xét nghiệm (máu, nước tiểu, chụp X-quang, siêu âm,...)
Điều kiện hậu quả	Xét nghiệm hoàn thành và chuẩn bị cho việc nhập kết quả vào hệ thống.

3. Use Case: Nhập kết quả xét nghiệm lên hệ thống (có thể kèm hình ảnh, file đính kèm)



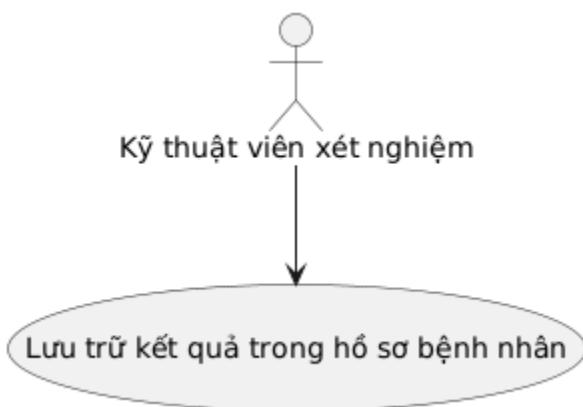
Tên Use Case	Nhập kết quả xét nghiệm lên hệ thống (có thể kèm hình ảnh, file đính kèm)
Mô tả	Kỹ thuật viên nhập kết quả xét nghiệm vào hệ thống, có thể bao gồm các hình ảnh (X-quang, siêu âm) và các file đính kèm (ví dụ, kết quả xét nghiệm máu).
Diễn viên	Kỹ thuật viên xét nghiệm
Luồng sự kiện chính	<ol style="list-style-type: none"> Kỹ thuật viên hoàn thành xét nghiệm. Kỹ thuật viên nhập kết quả vào hệ thống, kèm theo các hình ảnh và tài liệu cần thiết. Hệ thống lưu trữ kết quả và thông báo cho bác sĩ khi có kết quả.
Điều kiện tiên quyết	Kết quả xét nghiệm đã được hoàn thành và có thể nhập vào hệ thống.
Điều kiện hậu quả	Kết quả xét nghiệm được lưu trữ trong hồ sơ bệnh nhân và sẵn sàng để bác sĩ tham khảo.

4. Use Case: Gửi thông báo cho bác sĩ khi có kết quả



Tên Use Case	Gửi thông báo cho bác sĩ khi có kết quả
Mô tả	Kỹ thuật viên gửi thông báo tới bác sĩ khi kết quả xét nghiệm đã được nhập vào hệ thống.
Diễn viên	Kỹ thuật viên xét nghiệm, Bác sĩ
Luồng sự kiện chính	<ol style="list-style-type: none"> Kỹ thuật viên nhập kết quả xét nghiệm vào hệ thống. Hệ thống tự động gửi thông báo (email/SMS) đến bác sĩ. Bác sĩ nhận thông báo và có thể truy cập kết quả xét nghiệm để đánh giá.
Điều kiện tiên quyết	Kết quả xét nghiệm đã được nhập vào hệ thống và bác sĩ đã đăng ký nhận thông báo.
Điều kiện hậu quả	Bác sĩ nhận thông báo và có thể tiếp tục quá trình điều trị cho bệnh nhân.

5. Use Case: Lưu trữ kết quả trong hồ sơ bệnh nhân



Tên Use Case	Lưu trữ kết quả trong hồ sơ bệnh nhân
Mô tả	Kỹ thuật viên lưu trữ kết quả xét nghiệm vào hồ sơ bệnh nhân trong hệ thống, để bác sĩ và các nhân viên y tế khác có thể tham khảo.
Diễn viên	Kỹ thuật viên xét nghiệm
Luồng sự kiện chính	<ol style="list-style-type: none"> Kỹ thuật viên hoàn thành việc xét nghiệm và nhập kết quả vào hệ thống. Hệ thống tự động lưu kết quả vào hồ sơ bệnh nhân. Kết quả có sẵn cho bác sĩ, y tá và các nhân viên y tế khác.
Điều kiện tiên quyết	Kết quả xét nghiệm đã được hoàn thành và hệ thống có khả năng lưu trữ thông tin.
Điều kiện hậu quả	Kết quả được lưu trữ an toàn và có thể truy cập bởi các nhân viên y tế khi cần thiết.

1.2 Analyze Requirements

1.2.1 Decompose the system in microservices with Django

1. Giới thiệu về Microservices

Kiến trúc **microservices** là một cách tiếp cận phát triển hệ thống phân tán, trong đó các ứng dụng được chia thành các dịch vụ độc lập, nhỏ gọn, mỗi dịch vụ chỉ đảm nhận một phần nghiệp vụ cụ thể. Điều này giúp hệ thống dễ dàng mở rộng, bảo trì và có thể triển khai các phần riêng biệt mà không cần phải thay đổi toàn bộ hệ thống.

Lý do chọn microservices cho hệ thống **Health-Care**:

- Tính mở rộng:** Mỗi dịch vụ có thể mở rộng độc lập với các dịch vụ khác khi có nhu cầu.
- Tính dễ bảo trì:** Các dịch vụ độc lập giúp dễ dàng cập nhật, bảo trì mà không ảnh hưởng đến các phần còn lại của hệ thống.
- Tính khả dụng cao:** Khi một dịch vụ gặp sự cố, các dịch vụ khác vẫn hoạt động bình thường mà không bị ảnh hưởng.

2. Phân tách hệ thống thành các microservices

Hệ thống **Health-Care** sẽ được chia thành các dịch vụ sau:

User Management Service (Dịch vụ Quản lý Người dùng)

- **Chức năng chính:**

- Đăng ký tài khoản cho bệnh nhân, bác sĩ, y tá, dược sĩ, quản trị viên.
- Đăng nhập vào hệ thống.
- Cập nhật thông tin cá nhân của người dùng.

- **API:**

- Đăng ký tài khoản (POST /register).
- Đăng nhập (POST /login).
- Cập nhật thông tin cá nhân (PUT /update-profile).

Appointment Service (Dịch vụ Quản lý Lịch hẹn)

- **Chức năng chính:**

- Đặt lịch khám cho bệnh nhân với bác sĩ.
- Xem lịch khám của bệnh nhân.
- Hủy lịch hẹn.

- **API:**

- Đặt lịch hẹn (POST /appointments).
- Xem lịch hẹn (GET /appointments).
- Hủy lịch hẹn (DELETE /appointments/{id}).

Medical Record Service (Dịch vụ Hồ sơ Bệnh án)

- **Chức năng chính:**

- Lưu trữ và truy xuất hồ sơ bệnh án của bệnh nhân.
- Cập nhật thông tin hồ sơ bệnh án sau mỗi lần khám.

- **API:**

- Lưu hồ sơ bệnh án (POST /medical-record).
- Xem hồ sơ bệnh án (GET /medical-record/{id}).

Prescription Service (Dịch vụ Quản lý Đơn thuốc)

- **Chức năng chính:**
 - Lập đơn thuốc cho bệnh nhân.
 - Theo dõi tình trạng cấp phát thuốc.
- **API:**
 - Lập đơn thuốc (POST /prescriptions).
 - Xem đơn thuốc (GET /prescriptions/{id}).

Lab Test Service (Dịch vụ Xét nghiệm)

- **Chức năng chính:**
 - Quản lý chỉ định xét nghiệm từ bác sĩ.
 - Nhập kết quả xét nghiệm vào hệ thống.
 - Gửi thông báo cho bác sĩ khi có kết quả xét nghiệm.
- **API:**
 - Chỉ định xét nghiệm (POST /lab-tests).
 - Nhập kết quả xét nghiệm (POST /lab-tests/results).
 - Gửi thông báo kết quả (POST /lab-tests/notify).

3. Công nghệ và Framework sử dụng

Hệ thống **Health-Care** sẽ sử dụng **Django** để triển khai các microservices, mỗi dịch vụ sẽ là một ứng dụng Django độc lập. Django sẽ cung cấp các tính năng mạnh mẽ để xây dựng ứng dụng web như:

- **Models:** Để quản lý các đối tượng cơ sở dữ liệu.
- **Views:** Để xử lý các yêu cầu HTTP và trả về kết quả.
- **URLs:** Để định tuyến các yêu cầu đến đúng các views.
- **Django REST Framework (DRF):** Để xây dựng các API RESTful cho các dịch vụ.

4. Cách các microservices giao tiếp với nhau

Mỗi microservice sẽ giao tiếp với các dịch vụ khác thông qua **REST APIs**. Ví dụ, khi bác sĩ chỉ định xét nghiệm cho bệnh nhân, **Lab Test Service** sẽ nhận chỉ định từ **Medical Record Service** hoặc **Appointment Service** thông qua các API tương ứng.

- **API Gateway** có thể được sử dụng để tập hợp tất cả các dịch vụ và làm cầu nối giữa client (bệnh nhân, bác sĩ, y tá) và các microservices.
- Các microservices có thể giao tiếp với nhau bằng các giao thức như **HTTP/REST**, **JSON** hoặc **gRPC** (nếu cần).

5. Triển khai và mở rộng các Microservices

Hệ thống sẽ sử dụng Docker để container hóa các microservices và Kubernetes để quản lý việc triển khai và mở rộng. Mỗi microservice sẽ được triển khai độc lập, và khi có nhu cầu mở rộng, chúng ta có thể mở rộng các microservice có tải cao mà không ảnh hưởng đến các phần khác của hệ thống.

Mỗi dịch vụ sẽ có cơ sở dữ liệu riêng biệt, giúp giảm sự phụ thuộc giữa các dịch vụ. Việc sử dụng cơ sở dữ liệu riêng giúp mỗi dịch vụ có thể tối ưu hóa cơ sở dữ liệu của mình mà không ảnh hưởng đến các dịch vụ khác.

1.2.2 Classes with attributes of service models (models)

Các model cho hệ thống:

1. Model: User (Người dùng)

- **Mô tả:** Lớp này đại diện cho tất cả các loại người dùng trong hệ thống, bao gồm bệnh nhân, bác sĩ, y tá, quản trị viên, dược sĩ, kỹ thuật viên xét nghiệm, và nhà cung cấp bảo hiểm.
- **Thuộc tính:**
 - id: Mã định danh người dùng (Primary Key)
 - username: Tên người dùng (Chuỗi)
 - password: Mật khẩu (Chuỗi)
 - email: Địa chỉ email (Chuỗi)
 - role: Vai trò của người dùng (Ví dụ: "patient", "doctor", "nurse", "admin", "pharmacist", "laboratory_technician", "insurance_provider") (Chuỗi)
 - created_at: Thời gian tạo tài khoản (Datetime)
 - updated_at: Thời gian cập nhật tài khoản (Datetime)

2. Model: Appointment (Lịch hẹn)

- **Mô tả:** Lớp này lưu trữ các cuộc hẹn giữa bệnh nhân và bác sĩ.
- **Thuộc tính:**
 - id: Mã cuộc hẹn (Primary Key)
 - patient_id: Mã bệnh nhân (ForeignKey liên kết với User)
 - doctor_id: Mã bác sĩ (ForeignKey liên kết với User)
 - appointment_time: Thời gian cuộc hẹn (Datetime)
 - status: Trạng thái cuộc hẹn (Ví dụ: "pending", "completed", "cancelled") (Chuỗi)

3. Model: MedicalRecord (Hồ sơ bệnh án)

- **Mô tả:** Lớp này lưu trữ các hồ sơ khám bệnh của bệnh nhân.
- **Thuộc tính:**
 - id: Mã hồ sơ bệnh án (Primary Key)
 - patient_id: Mã bệnh nhân (ForeignKey liên kết với User)
 - doctor_id: Mã bác sĩ (ForeignKey liên kết với User)
 - diagnosis: Chẩn đoán của bác sĩ (Chuỗi)
 - created_at: Thời gian tạo hồ sơ bệnh án (Datetime)

4. Model: Prescription (Đơn thuốc)

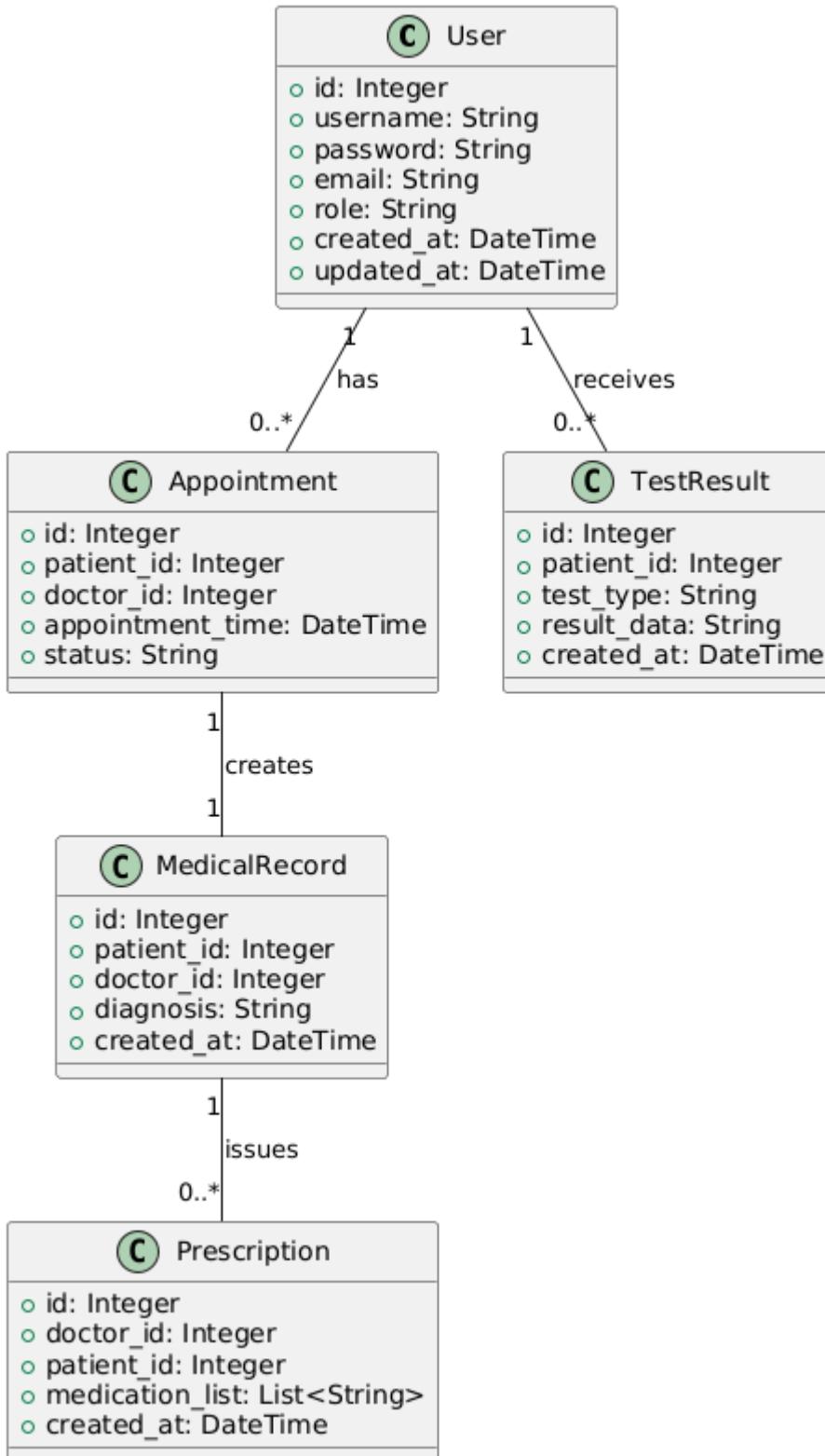
- **Mô tả:** Lớp này lưu trữ các đơn thuốc mà bác sĩ cấp cho bệnh nhân.
- **Thuộc tính:**
 - id: Mã đơn thuốc (Primary Key)
 - doctor_id: Mã bác sĩ (ForeignKey liên kết với User)
 - patient_id: Mã bệnh nhân (ForeignKey liên kết với User)
 - medication_list: Danh sách thuốc được kê (Danh sách chuỗi)
 - created_at: Thời gian cấp đơn thuốc (Datetime)

5. Model: TestResult (Kết quả xét nghiệm)

- **Mô tả:** Lớp này lưu trữ kết quả xét nghiệm của bệnh nhân.

- **Thuộc tính:**

- id: Mã kết quả xét nghiệm (Primary Key)
- patient_id: Mã bệnh nhân (ForeignKey liên kết với User)
- test_type: Loại xét nghiệm (Chuỗi, ví dụ: "blood test", "x-ray", "ultrasound")
- result_data: Dữ liệu kết quả xét nghiệm (Chuỗi hoặc JSON)
- created_at: Thời gian thực hiện xét nghiệm (Datetime)



Giải thích Biểu đồ:

- **User** (Người dùng) có thể có nhiều **Appointment** (Cuộc hẹn), **TestResult** (Kết quả xét nghiệm), và **Insurance** (Thông tin bảo hiểm).
- **Appointment** (Cuộc hẹn) có thể tạo ra một **MedicalRecord** (Hồ sơ bệnh án).
- **MedicalRecord** (Hồ sơ bệnh án) có thể có nhiều **Prescription** (Đơn thuốc).
- **User** có thể có nhiều **TestResult**, giúp hỗ trợ việc xét nghiệm.

1.2.3 Determine functions in services (views)

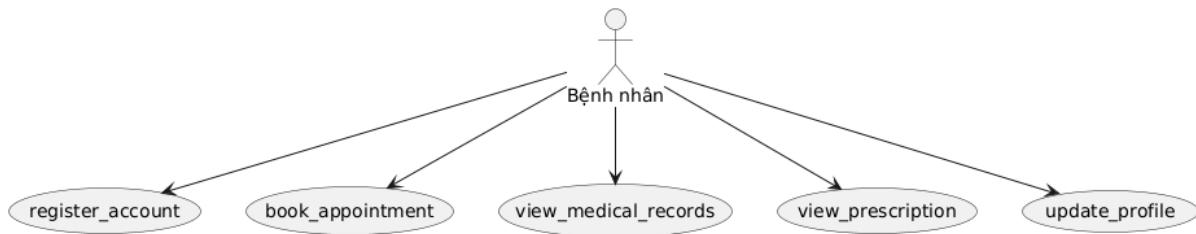
1. Bệnh nhân (Patient)

Use Cases:

- Đăng ký tài khoản
- Đặt lịch khám với bác sĩ
- Xem lịch sử khám chữa bệnh
- Xem đơn thuốc
- Cập nhật thông tin cá nhân

Các Views:

- **register_account(request)**: Đăng ký tài khoản mới cho bệnh nhân.
- **book_appointment(request)**: Đặt lịch khám với bác sĩ.
- **view_medical_records(request)**: Xem lịch sử khám chữa bệnh.
- **view_prescription(request)**: Xem đơn thuốc đã kê.
- **update_profile(request)**: Cập nhật thông tin cá nhân.



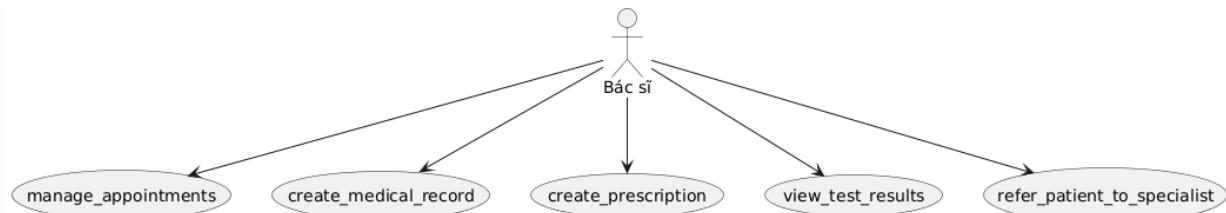
2. Bác sĩ (Doctor)

Use Cases:

- Quản lý lịch hẹn
- Chẩn đoán bệnh và tạo hồ sơ
- Kê đơn thuốc
- Xem kết quả xét nghiệm
- Giới thiệu bệnh nhân đến bác sĩ chuyên khoa

Các Views:

- **manage_appointments(request)**: Quản lý lịch hẹn của bệnh nhân.
- **create_medical_record(request)**: Chẩn đoán bệnh và tạo hồ sơ cho bệnh nhân.
- **create_prescription(request)**: Kê đơn thuốc cho bệnh nhân.
- **view_test_results(request)**: Xem kết quả xét nghiệm của bệnh nhân.
- **refer_patient_to_specialist(request)**: Giới thiệu bệnh nhân đến bác sĩ chuyên khoa.



3. Dược sĩ (Pharmacist)

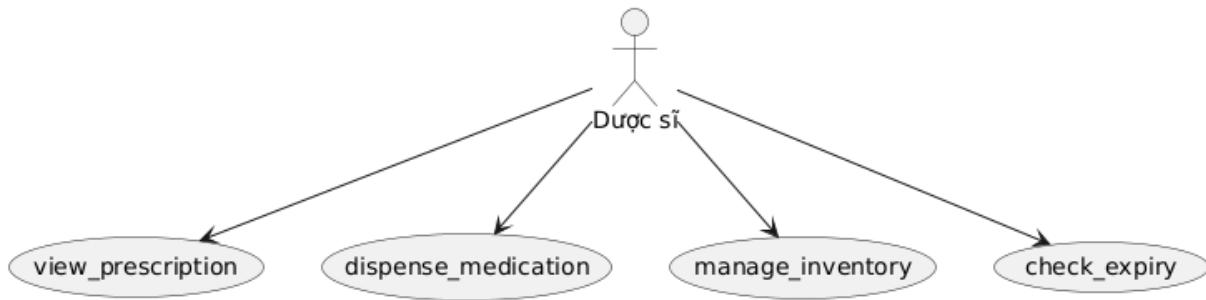
Use Cases:

- Xem đơn thuốc
- Cấp phát thuốc
- Quản lý tồn kho thuốc
- Kiểm tra hạn sử dụng thuốc

Các Views:

- **view_prescription(request)**: Xem đơn thuốc mà bác sĩ đã kê cho bệnh nhân.
- **dispense_medication(request)**: Cấp phát thuốc cho bệnh nhân theo đơn thuốc.

- **manage_inventory(request)**: Quản lý tồn kho thuốc.
- **check_expiry(request)**: Kiểm tra hạn sử dụng thuốc.



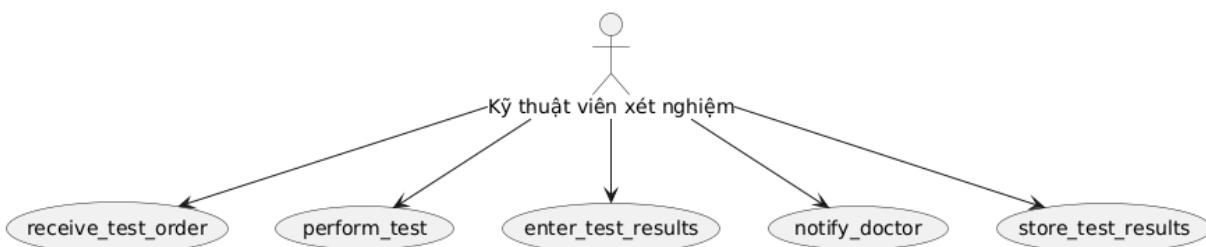
4. Kỹ thuật viên xét nghiệm (Laboratory Technician)

Use Cases:

- Nhận chỉ định xét nghiệm từ bác sĩ
- Thực hiện xét nghiệm
- Nhập kết quả xét nghiệm lên hệ thống
- Gửi thông báo cho bác sĩ khi có kết quả
- Lưu trữ kết quả trong hồ sơ bệnh nhân

Các Views:

- **receive_test_order(request)**: Nhận chỉ định xét nghiệm từ bác sĩ.
- **perform_test(request)**: Thực hiện xét nghiệm y tế.
- **enter_test_results(request)**: Nhập kết quả xét nghiệm lên hệ thống.
- **notify_doctor(request)**: Gửi thông báo cho bác sĩ khi có kết quả xét nghiệm.
- **store_test_results(request)**: Lưu trữ kết quả xét nghiệm vào hồ sơ bệnh nhân.



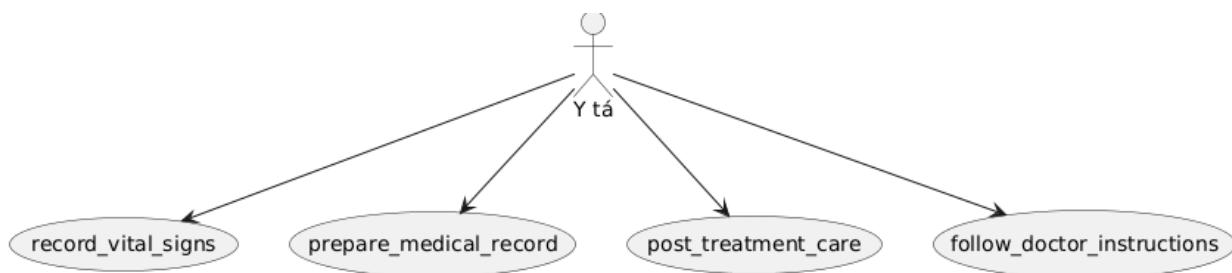
5. Y tá (Nurse)

Use Cases:

- Đo và ghi nhận các chỉ số sinh tồn
- Chuẩn bị hồ sơ bệnh nhân
- Chăm sóc bệnh nhân sau điều trị
- Thực hiện chỉ định từ bác sĩ

Các Views:

- **record_vital_signs(request)**: Đo và ghi nhận các chỉ số sinh tồn của bệnh nhân.
- **prepare_medical_record(request)**: Chuẩn bị hồ sơ bệnh nhân.
- **post_treatment_care(request)**: Chăm sóc bệnh nhân sau điều trị.
- **follow_doctor_instructions(request)**: Thực hiện chỉ định từ bác sĩ.



6. Quản trị viên (Administrator)

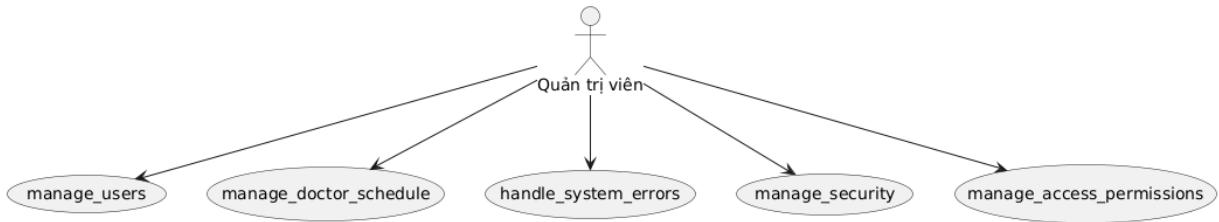
Use Cases:

- Quản lý người dùng hệ thống
- Quản lý lịch làm việc của bác sĩ
- Xử lý lỗi hệ thống
- Bảo mật hệ thống
- Quản lý quyền truy cập

Các Views:

- **manage_users(request)**: Quản lý người dùng hệ thống.
- **manage_doctor_schedule(request)**: Quản lý lịch làm việc của bác sĩ.
- **handle_system_errors(request)**: Xử lý các lỗi hệ thống.

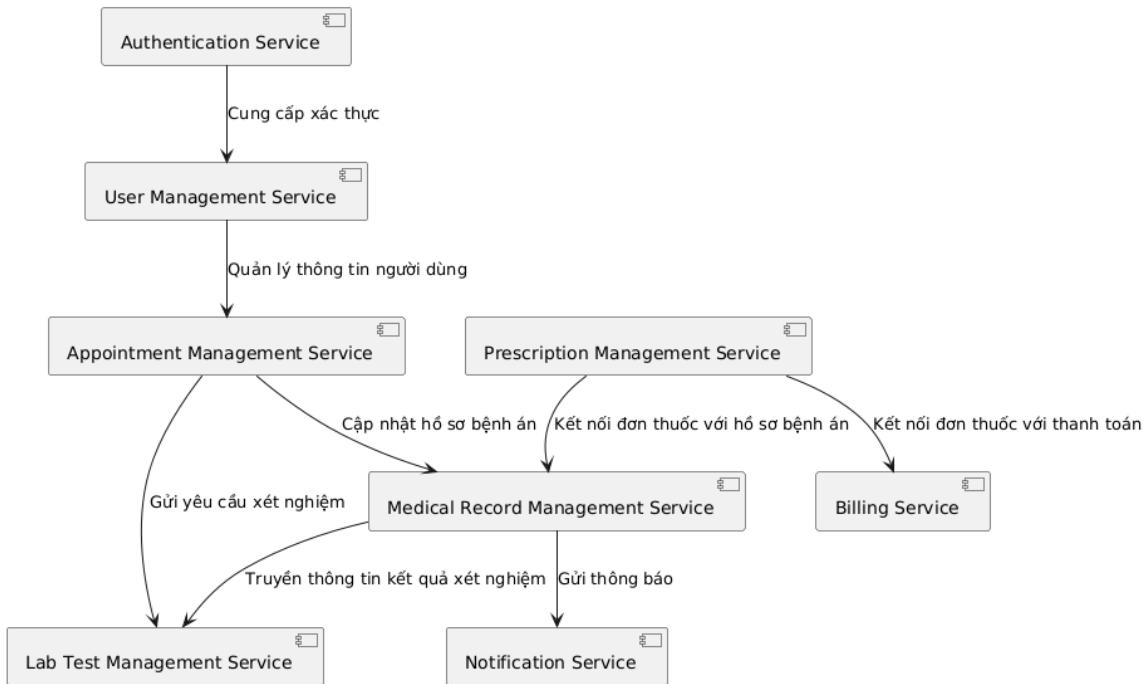
- **manage_security(request)**: Quản lý bảo mật hệ thống.
- **manage_access_permissions(request)**: Quản lý quyền truy cập.



1.2.5 Determine REST API connecting services

Các Service chính trong hệ thống:

Dưới đây là các **service** chính trong hệ thống và các **API endpoints** tương ứng cho từng service, kèm theo biểu đồ PlantUML mô tả các kết nối giữa các service này.



Các Service trong Hệ thống:

1. User Management Service (Dịch vụ Quản lý Người dùng)

- **Chức năng:** Quản lý tài khoản người dùng (bệnh nhân, bác sĩ, dược sĩ, v.v.), đăng ký, đăng nhập, phân quyền.
- **API Endpoints:**
 - POST /users/register: Đăng ký tài khoản.
 - POST /users/login: Đăng nhập.
 - GET /users/{id}: Lấy thông tin người dùng.
 - PUT /users/{id}: Cập nhật thông tin người dùng.

2. Appointment Service (Dịch vụ Quản lý Lịch hẹn)

- **Chức năng:** Quản lý lịch hẹn giữa bệnh nhân và bác sĩ.
- **API Endpoints:**
 - POST /appointments: Tạo lịch hẹn.
 - GET /appointments/{id}: Lấy thông tin lịch hẹn.
 - PUT /appointments/{id}: Cập nhật lịch hẹn.
 - DELETE /appointments/{id}: Hủy lịch hẹn.

3. Medical Record Service (Dịch vụ Hồ sơ Bệnh án)

- **Chức năng:** Quản lý và lưu trữ hồ sơ bệnh án của bệnh nhân.
- **API Endpoints:**
 - GET /medical-records/{patientId}: Lấy hồ sơ bệnh án của bệnh nhân.
 - POST /medical-records: Thêm hồ sơ bệnh án mới.
 - PUT /medical-records/{id}: Cập nhật hồ sơ bệnh án.

4. Prescription Service (Dịch vụ Quản lý Đơn thuốc)

- **Chức năng:** Quản lý đơn thuốc của bệnh nhân.
- **API Endpoints:**
 - POST /prescriptions: Tạo đơn thuốc mới.
 - GET /prescriptions/{id}: Lấy thông tin đơn thuốc.
 - PUT /prescriptions/{id}: Cập nhật đơn thuốc.

- DELETE /prescriptions/{id}: Hủy đơn thuốc.

5. Lab Test Service (Dịch vụ Xét nghiệm)

- **Chức năng:** Quản lý xét nghiệm y tế, kết quả xét nghiệm.
- **API Endpoints:**
 - POST /lab-tests: Tạo yêu cầu xét nghiệm.
 - GET /lab-tests/{id}: Lấy kết quả xét nghiệm.
 - PUT /lab-tests/{id}: Cập nhật kết quả xét nghiệm.

6. Auth Service (Dịch vụ Xác thực)

- **Chức năng:** Xác thực người dùng, quản lý đăng nhập, đăng ký và phân quyền.
- **API Endpoints:**
 - POST /auth/login: Đăng nhập người dùng.
 - POST /auth/register: Đăng ký người dùng.
 - POST /auth/logout: Đăng xuất người dùng.
 - GET /auth/refresh-token: Làm mới token xác thực.

7. Notification Service (Dịch vụ Thông báo)

- **Chức năng:** Gửi thông báo tới người dùng.
- **API Endpoints:**
 - POST /notifications: Gửi thông báo.
 - GET /notifications/{userId}: Lấy thông báo của người dùng.

8. Billing Service (Dịch vụ Thanh toán)

- **Chức năng:** Quản lý các hóa đơn, thanh toán cho các dịch vụ y tế.
- **API Endpoints:**
 - POST /billing/invoice: Tạo hóa đơn.
 - GET /billing/{invoiceId}: Lấy thông tin hóa đơn.
 - POST /billing/payment: Thanh toán hóa đơn.

1.3 Conclusion

Trong phần phân tích yêu cầu (Requirements Analysis) cho hệ thống chăm sóc sức khỏe, chúng ta đã tập trung vào việc xác định các yêu cầu cơ bản của hệ thống từ các actor khác nhau, bao gồm bệnh nhân, bác sĩ, y tá, dược sĩ, kỹ thuật viên xét nghiệm, quản trị viên và các tác nhân khác như nhà cung cấp bảo hiểm. Mỗi actor có những yêu cầu chức năng và phi chức năng riêng, đóng vai trò quan trọng trong việc cung cấp dịch vụ chăm sóc sức khỏe.

Dựa trên các use case, chúng ta đã phân tích và xác định các dịch vụ cần thiết trong hệ thống, chẳng hạn như dịch vụ quản lý người dùng, dịch vụ lịch hẹn, dịch vụ đơn thuốc, dịch vụ xét nghiệm, và dịch vụ thanh toán. Các dịch vụ này cần được phân chia thành các microservices độc lập, mỗi microservice sẽ đảm nhận một chức năng riêng biệt, giảm thiểu sự phụ thuộc giữa các phần trong hệ thống và cải thiện khả năng mở rộng cũng như bảo trì hệ thống.

Trong quá trình này, chúng ta cũng đã xác định các model dữ liệu chính (như Patient, Doctor, Prescription, Appointment, v.v.) và các function/views liên quan đến từng actor. Điều này giúp đảm bảo rằng các yêu cầu về mặt nghiệp vụ sẽ được triển khai một cách chính xác và rõ ràng, đồng thời duy trì tính dễ sử dụng cho người dùng cuối.

Ngoài ra, việc xây dựng một kiến trúc RESTful API cho phép các microservices giao tiếp với nhau một cách dễ dàng và hiệu quả. Mỗi service sẽ tương tác qua các endpoint RESTful, giúp hệ thống linh hoạt trong việc tích hợp và mở rộng sau này.

Tóm lại, việc phân tích các yêu cầu cho hệ thống chăm sóc sức khỏe là bước đầu tiên quan trọng trong việc thiết kế và triển khai một hệ thống hiệu quả. Các yêu cầu đã được phân tích và chi tiết hóa, giúp đảm bảo rằng hệ thống sẽ đáp ứng được nhu cầu của tất cả các actor và chức năng nghiệp vụ, đồng thời duy trì tính bền vững và mở rộng trong tương lai.

CHAPTER 2: DESIGN E-COMMERCE SYSTEM WITH MICROSERVICES AND DJANGO

2.1.1 System Overview (Tổng quan hệ thống)

Hệ thống E-commerce được xây dựng theo kiến trúc **Microservices** sử dụng các công nghệ **Django**, **Angular**, và các cơ sở dữ liệu **PostgreSQL**, **MySQL**, **MongoDB**, cùng với **Redis** cho caching và **Nginx** làm API Gateway. Kiến trúc này giúp đảm bảo tính mở rộng, linh hoạt và khả năng quản lý độc lập cho từng dịch vụ trong hệ thống.

Các thành phần chính trong hệ thống:

1. Frontend (Angular):

- Là phần giao diện người dùng của hệ thống.
- Sử dụng **Angular** để xây dựng các ứng dụng web tương tác với người dùng. Các tính năng như đăng ký, đăng nhập, quản lý giỏ hàng, thanh toán và theo dõi đơn hàng đều sẽ được người dùng thực hiện trên giao diện này.

2. API Gateway (Nginx):

- **Nginx** sẽ đóng vai trò là API Gateway, điều hướng các yêu cầu từ frontend đến các microservices khác nhau.
- Ngoài việc định tuyến, **Nginx** còn xử lý các vấn đề như xác thực, cân bằng tải, và giới hạn tốc độ (rate-limiting) để đảm bảo an toàn và hiệu suất hệ thống.

3. Backend (Microservices với Django):

- Các microservices được xây dựng với Django, mỗi service sẽ có một **Django app riêng biệt**, hoạt động độc lập và có cơ sở dữ liệu riêng.
- Backend sẽ bao gồm các dịch vụ như: **User Service**, **Product Service**, **Order Service**, **Payment Service**, và **Shipping Service**. Mỗi service sẽ xử lý một phần của quy trình e-commerce.

4. Database:

- **PostgreSQL** và **MySQL** được sử dụng tùy theo nhu cầu của từng service.
PostgreSQL sẽ được sử dụng cho các service có nhu cầu ghi/đọc dữ liệu phức tạp (như Order Service), trong khi **MySQL** phù hợp với các service yêu cầu nhiều truy vấn đọc, ít cập nhật (như Product Service).
- **MongoDB** sẽ được sử dụng cho các trường hợp dữ liệu không có cấu trúc hoặc cần tính mở rộng cao, ví dụ như **User Service** để lưu trữ thông tin người dùng có thể thay đổi thường xuyên.

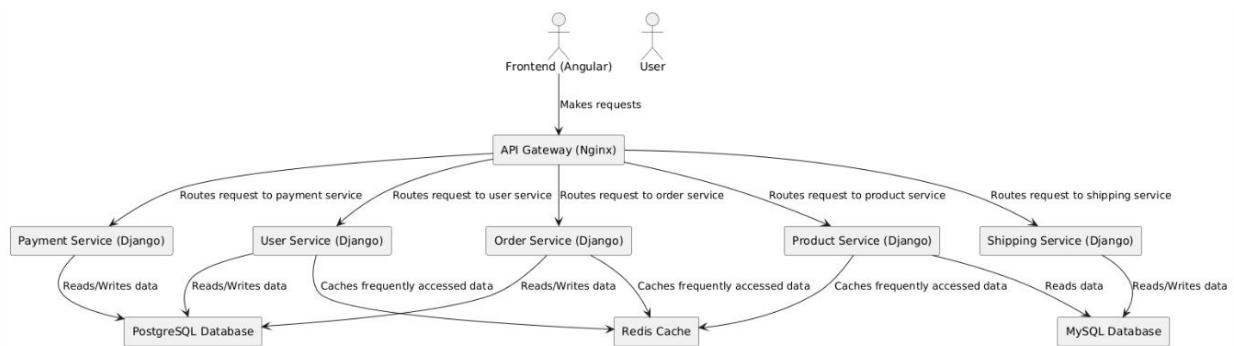
5. Cache (Redis):

- Redis sẽ được sử dụng để cache các dữ liệu cần truy cập nhanh như thông tin sản phẩm, thông tin người dùng, hoặc các kết quả tìm kiếm sản phẩm.
- Điều này giúp giảm tải cho database và cải thiện hiệu suất hệ thống.

Các tính năng của hệ thống:

- **Scalability (Khả năng mở rộng):** Với kiến trúc microservices, các service có thể mở rộng độc lập và dễ dàng bổ sung thêm service mới khi có yêu cầu.
- **High Availability (Khả năng sẵn sàng cao):** Các dịch vụ có thể được triển khai trên nhiều máy chủ, đảm bảo rằng hệ thống luôn sẵn sàng hoạt động ngay cả khi một hoặc nhiều phần của hệ thống gặp sự cố.
- **Security (Bảo mật):** API Gateway sẽ xác thực các yêu cầu từ frontend, chỉ cho phép các yêu cầu hợp lệ được chuyển đến các dịch vụ backend. Các token JWT sẽ được sử dụng để đảm bảo bảo mật.
- **Resilience (Khả năng phục hồi):** Microservices độc lập giúp hệ thống dễ dàng phục hồi nếu một service gặp sự cố mà không ảnh hưởng đến toàn bộ hệ thống.

Sơ đồ Kiến Trúc Tổng Thể



2.1.2 Technology Stack

Hệ thống thương mại điện tử được xây dựng theo kiến trúc microservices, sử dụng các công nghệ hiện đại nhằm đảm bảo khả năng mở rộng, bảo trì, và hiệu năng cao. Dưới đây là mô tả chi tiết các công nghệ được sử dụng cho từng thành phần chính trong hệ thống:

Backend

- **Django:** Mỗi service trong hệ thống được hiện thực như một Django app độc lập. Django cung cấp bộ công cụ mạnh mẽ giúp xây dựng nhanh các ứng dụng web RESTful, hỗ trợ ORM và tích hợp bảo mật tốt.

- **Django REST Framework (DRF)**: Được sử dụng để xây dựng REST API cho các service, hỗ trợ serialization, validation và các phương thức HTTP.

Frontend

- **Angular**: Là framework frontend chính, cung cấp giao diện người dùng hiện đại, tương tác cao. Angular giao tiếp với backend thông qua các REST API và được triển khai tách biệt như một ứng dụng SPA (Single Page Application).

Database

- **PostgreSQL**: Được sử dụng cho các service có yêu cầu ghi dữ liệu thường xuyên và cần tính toàn vẹn dữ liệu cao (ví dụ: Order Service, Payment Service).
- **MySQL**: Được sử dụng cho các service chủ yếu thực hiện thao tác đọc, nơi yêu cầu tốc độ truy vấn cao (ví dụ: Product Service, Shipping Service).
- **MongoDB**: Được áp dụng cho các service có cấu trúc dữ liệu linh hoạt và dễ thay đổi (ví dụ: User Service), nhằm tận dụng khả năng mở rộng theo chiều ngang và lưu trữ dữ liệu phi quan hệ.

Communication

- **RESTful API**: Các service giao tiếp với nhau và với frontend thông qua giao thức HTTP/HTTPS theo chuẩn REST, đảm bảo tính thống nhất và dễ tích hợp.

API Gateway

- **Nginx**: Được sử dụng làm API Gateway để định tuyến các request đến các service tương ứng. Ngoài ra, Nginx cũng xử lý load balancing, caching tĩnh và bảo mật (SSL termination, rate limiting...).

Cache

- **Redis**: Được dùng để cache các dữ liệu được truy cập thường xuyên nhằm giảm tải cho hệ thống backend và tăng tốc độ phản hồi. Redis có thể được sử dụng để lưu trữ session, giỏ hàng, hay kết quả các truy vấn phổ biến.

Containerization & Orchestration

- **Docker**: Mỗi service được đóng gói như một container độc lập, giúp đảm bảo tính đồng nhất trong quá trình phát triển và triển khai.
- **Docker Compose**: Hệ thống sử dụng Docker Compose để định nghĩa và quản lý toàn bộ các container trong môi trường local, cho phép khởi chạy nhiều service cùng lúc với cấu hình đơn giản.

2.1.3 Microservices Breakdown (Các dịch vụ chính trong hệ thống)

Hệ thống được thiết kế theo kiến trúc microservices nhằm đảm bảo tính linh hoạt, khả năng mở rộng, dễ dàng bảo trì và phân tách trách nhiệm rõ ràng giữa các thành phần. Dựa trên phân tích tác nhân và các chức năng (use case) của hệ thống, các microservices chính được đề xuất như sau:

1. User Service

Chịu trách nhiệm xử lý thông tin người dùng, bao gồm bệnh nhân, bác sĩ, y tá, dược sĩ, kỹ thuật viên, và quản trị viên.

- Đăng ký tài khoản, đăng nhập OTP/email/mật khẩu.
- Cập nhật hồ sơ cá nhân, thông tin liên hệ, BHYT.
- Phân quyền và xác thực người dùng (authentication & authorization).
- Quản lý phiên đăng nhập và bảo mật người dùng.

2. Appointment Service

Xử lý toàn bộ luồng đặt lịch khám bệnh của bệnh nhân.

- Tìm kiếm bác sĩ theo chuyên khoa và lịch làm việc.
- Đặt lịch khám (trực tiếp hoặc trực tuyến).
- Quản lý lịch khám của bác sĩ.
- Gửi thông báo xác nhận lịch khám và nhắc lịch tự động.

3. Medical Record Service

Lưu trữ và quản lý hồ sơ bệnh án của bệnh nhân.

- Ghi nhận thông tin khám, chẩn đoán, ghi chú lâm sàng từ bác sĩ.
- Cập nhật thông tin lâm sàng từ y tá.
- Tạo báo cáo bệnh án điện tử.
- Lưu trữ lịch sử khám và truy xuất hồ sơ bệnh án.

4. Prescription Service

Quản lý đơn thuốc và chỉ định cận lâm sàng của bác sĩ.

- Tạo đơn thuốc, chỉnh sửa, gửi tới dược sĩ.

- Quản lý trạng thái đơn thuốc (đã cấp phát, hết thuốc, thay thế).
- Lưu trữ lịch sử kê đơn.

5. Pharmacy Service

Quản lý kho thuốc và hoạt động của dược sĩ.

- Xác nhận đơn thuốc và xử lý cấp phát thuốc.
- Quản lý tồn kho, nhập xuất thuốc, lô thuốc, hạn sử dụng.
- In phiếu thuốc và hóa đơn cấp phát.

6. Laboratory Service

Hỗ trợ kỹ thuật viên xét nghiệm trong quản lý quy trình xét nghiệm.

- Nhận chỉ định xét nghiệm từ bác sĩ.
- Nhập kết quả xét nghiệm, upload hình ảnh liên quan.
- Gửi kết quả xét nghiệm đến Medical Record Service.

7. Billing Service

Quản lý thanh toán dịch vụ và hóa đơn khám chữa bệnh.

- Tính toán chi phí theo dịch vụ sử dụng.
- Hỗ trợ thanh toán trực tuyến.
- Tạo và lưu trữ hóa đơn khám chữa bệnh.

8. Notification Service

Gửi thông báo hệ thống qua email hoặc SMS cho người dùng.

- Nhắc lịch khám, gửi kết quả xét nghiệm, đơn thuốc.
- Gửi phản hồi từ bác sĩ hoặc hệ thống.
- Giao tiếp thông qua hàng đợi nội bộ hoặc webhook (nếu mở rộng sau này).

9. Insurance Service

Quản lý quá trình bảo hiểm y tế của bệnh nhân.

- Gửi yêu cầu bảo hiểm, kiểm tra trạng thái xử lý.
- Theo dõi tiến trình duyệt bảo hiểm.

- Tích hợp với hệ thống bảo hiểm bên ngoài (nếu có).

10. Admin Service

Hỗ trợ quản trị hệ thống và người dùng nội bộ.

- Quản lý tài khoản và phân quyền người dùng.
- Quản lý chuyên khoa, phòng khám, dịch vụ.
- Theo dõi log hệ thống và các vấn đề kỹ thuật.

2.1.4 Service Communication (Giao tiếp giữa các service)

Hệ thống được triển khai theo kiến trúc microservices với các dịch vụ độc lập, mỗi dịch vụ chạy trong một container riêng biệt. Các container này nằm trong cùng một mạng Docker (docker network), cho phép các service giao tiếp nội bộ thông qua REST API với mô hình **request-response**.

Mô hình giao tiếp:

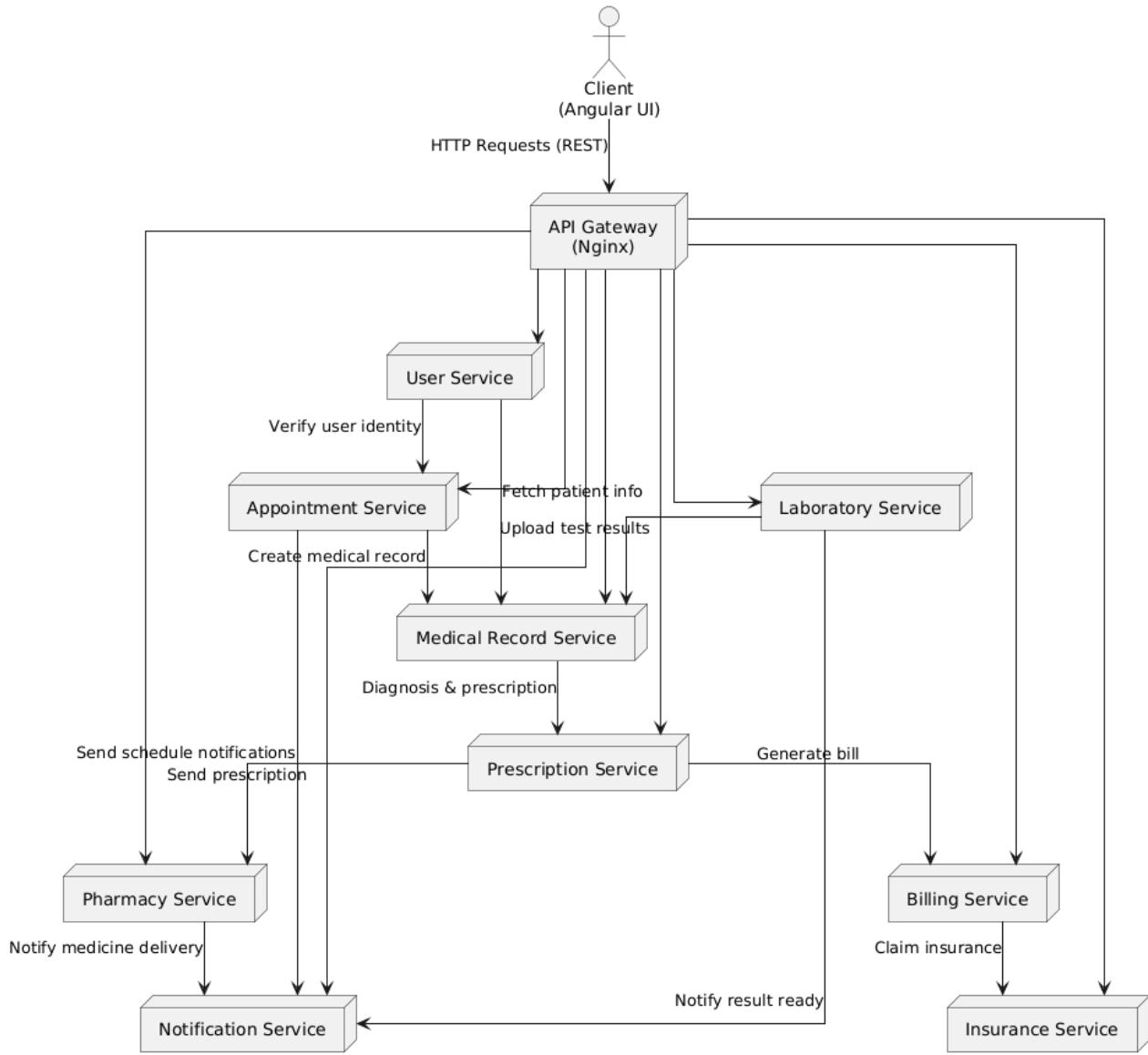
- **API Gateway (Nginx)** đóng vai trò là cổng truy cập duy nhất cho tất cả client (web, mobile). Gateway chịu trách nhiệm:
 - Định tuyến yêu cầu đến các service nội bộ phù hợp.
 - Chuyển tiếp phản hồi từ service về cho client.
 - Hỗ trợ logging, rate limiting, CORS, kiểm soát truy cập cơ bản.
- **Các service nội bộ giao tiếp với nhau trực tiếp** qua REST API sử dụng **hostname nội bộ** trong Docker Compose (ví dụ: `http://user-service:8000/api/...`).
- Dữ liệu được truyền theo chiều cụ thể dựa trên **luồng nghiệp vụ và trách nhiệm chức năng**.

Các nguyên tắc giao tiếp:

1. **Không sử dụng message broker:** toàn bộ các luồng trao đổi là đồng bộ (REST).
 2. **Chỉ có API Gateway tương tác với client bên ngoài,** các service không public ra ngoài.
 3. **Mỗi service chỉ gọi các service cần thiết,** tránh vòng lặp không cần thiết.
-

Quan hệ và chiều giao tiếp giữa các service

Source Service	Target Service	Lý do nghiệp vụ
API Gateway	Tất cả các service	Nhận request từ client, định tuyến đến service phù hợp
User Service	Appointment Service	Xác minh danh tính và lấy thông tin người dùng để đặt lịch
User Service	Medical Record Service	Gắn hồ sơ bệnh án theo người dùng
Appointment Service	Medical Record Service	Khởi tạo hồ sơ khám bệnh khi lịch hẹn bắt đầu
Appointment Service	Notification Service	Gửi thông báo xác nhận lịch hẹn và nhắc lịch
Medical Record Service	Prescription Service	Gửi thông tin chẩn đoán và yêu cầu kê đơn thuốc
Prescription Service	Pharmacy Service	Gửi đơn thuốc cho dược sĩ xử lý
Pharmacy Service	Notification Service	Gửi thông báo cấp phát thuốc thành công hoặc lỗi
Prescription Service	Billing Service	Tạo hóa đơn dựa trên đơn thuốc và dịch vụ
Billing Service	Insurance Service	Gửi thông tin yêu cầu thanh toán bảo hiểm
Laboratory Service	Medical Record Service	Trả kết quả xét nghiệm, cập nhật vào hồ sơ bệnh án
Laboratory Service	Notification Service	Gửi thông báo khi có kết quả mới
Doctor/Nurse UI	Giao tiếp qua API Gateway	Tương tác với các service thông qua API Gateway



2.1.5 Database Strategy (Chiến lược quản lý cơ sở dữ liệu)

Nguyên tắc thiết kế:

Hệ thống tuân thủ **nguyên tắc database-per-service**, tức là:

- **Mỗi service có database riêng**, được quản lý độc lập, **không chia sẻ schema, bảng**, hay **kết nối trực tiếp** giữa các service.
- **Tương tác liên service chỉ diễn ra qua REST API**, đảm bảo tính phân mảnh, dễ mở rộng, và tránh coupling.

- Dữ liệu mỗi service được thiết kế tối ưu theo **loại dữ liệu, tần suất đọc/ghi, tính phức tạp mối quan hệ**.
-

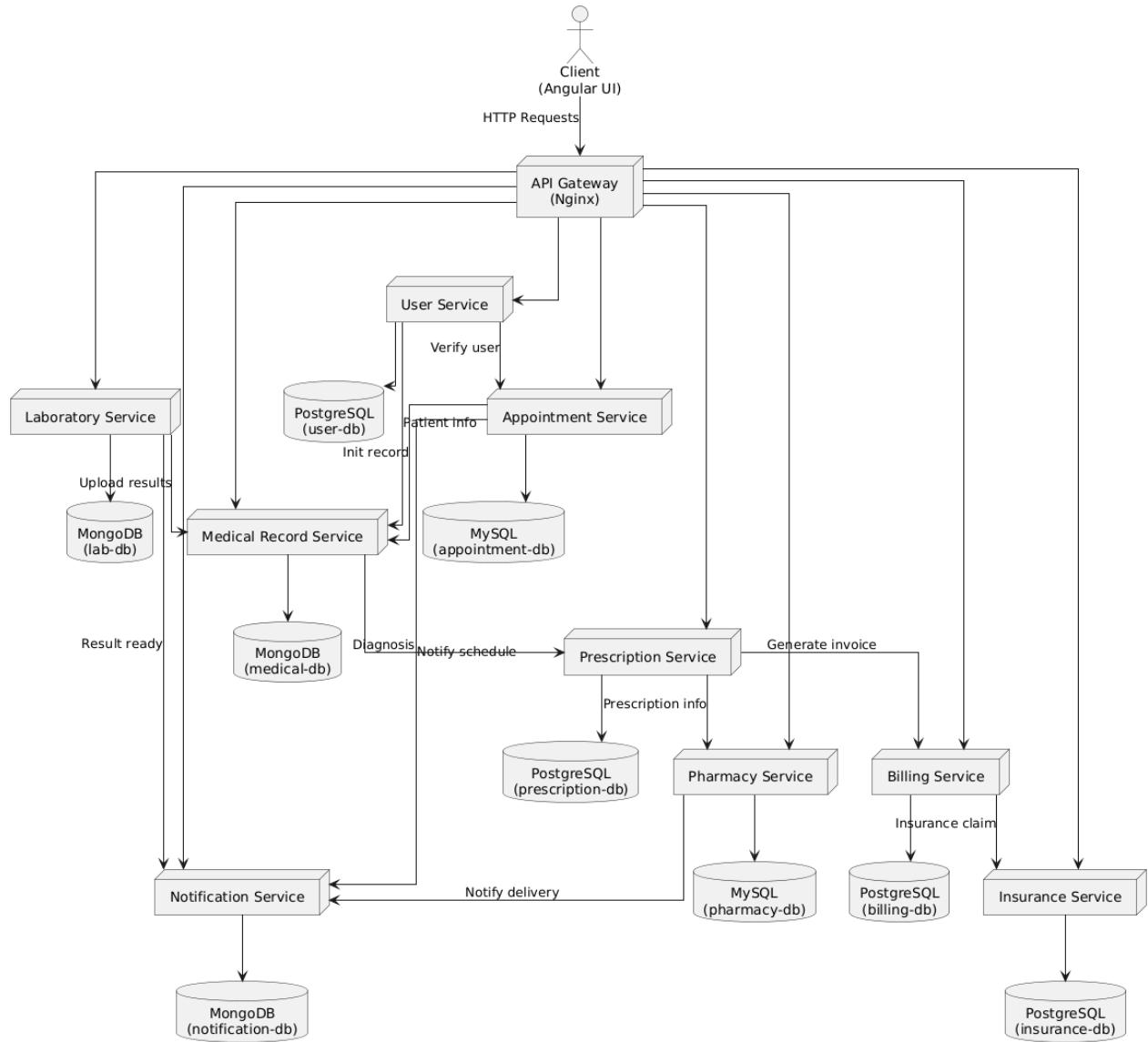
Lựa chọn cơ sở dữ liệu:

Service	Cơ sở dữ liệu	Lý do lựa chọn
User Service	PostgreSQL	Dữ liệu người dùng cần tính toàn vẹn cao, dùng các ràng buộc, mối quan hệ phức tạp.
Appointment Service	MySQL	Cấu trúc dữ liệu đơn giản, đọc/ghi liên tục, phù hợp với hiệu năng ổn định của MySQL.
Medical Record Service	MongoDB	Hồ sơ bệnh án có cấu trúc linh hoạt (lâm sàng, ảnh, đơn thuốc), rất phù hợp NoSQL.
Prescription Service	PostgreSQL	Có mối quan hệ giữa thuốc, chỉ định, đơn thuốc – cần ràng buộc khóa ngoại và tính toàn vẹn.
Pharmacy Service	MySQL	Quản lý kho thuốc và cấp phát đơn giản, transactional và dễ scale.
Billing Service	PostgreSQL	Giao dịch tài chính yêu cầu độ chính xác cao, truy vấn phức tạp, nhiều constraint.
Notification Service	MongoDB	Thông báo dạng log, cấu trúc không đồng nhất, yêu cầu lưu trữ linh hoạt, tốc độ cao.
Insurance Service	PostgreSQL	Lưu thông tin yêu cầu bảo hiểm, trạng thái xử lý, cần xác thực, phức tạp logic.
Laboratory Service	MongoDB	Kết quả xét nghiệm đa dạng (ảnh, PDF, mô tả...), dữ liệu phi cấu trúc, thích hợp NoSQL.

Tổng hợp:

- PostgreSQL (4 service):** User, Prescription, Billing, Insurance.
- MySQL (2 service):** Appointment, Pharmacy.

- **MongoDB (3 service):** Medical Record, Notification, Laboratory.



2.1.6 Security Considerations (Xem xét bảo mật)

Tổng quan:

Bảo mật là một yếu tố quan trọng trong việc thiết kế và triển khai hệ thống phân tán với nhiều dịch vụ như trong mô hình microservices. Để đảm bảo tính an toàn của dữ liệu người dùng và bảo vệ các API khỏi các cuộc tấn công không mong muốn, chúng ta sẽ triển khai cơ chế xác thực và phân quyền dựa trên **JWT (JSON Web Token)**. JWT sẽ giúp chúng ta duy trì các phiên làm việc an toàn và đảm bảo rằng các yêu cầu API chỉ có thể được thực hiện bởi các người dùng hợp lệ.

Cơ chế bảo mật với JWT:

1. Xác thực người dùng (Authentication):

- **Quá trình đăng nhập:** Khi người dùng (bệnh nhân, bác sĩ, quản trị viên, v.v.) đăng nhập vào hệ thống, hệ thống sẽ yêu cầu họ cung cấp thông tin xác thực (email và mật khẩu hoặc OTP).
- Sau khi xác thực thành công, **JWT** sẽ được tạo ra và trả về cho người dùng.
- JWT sẽ chứa các thông tin như **user ID**, **roles**, và **expiration time** để hệ thống có thể xác định quyền truy cập của người dùng và đảm bảo token không hết hạn.

2. Xác thực token (Token Validation):

- Mỗi lần người dùng gửi yêu cầu đến API, **JWT** sẽ được kèm theo trong phần header của yêu cầu dưới dạng Authorization: Bearer <token>.
- API Gateway (Nginx) hoặc các service sẽ xác thực token qua việc kiểm tra tính hợp lệ của nó bằng cách giải mã token, xác thực chữ ký và kiểm tra thời gian hết hạn.
- Nếu token hợp lệ, yêu cầu sẽ được chuyển tiếp đến service tương ứng. Nếu không hợp lệ (hết hạn hoặc bị thay đổi), một lỗi sẽ được trả về cho người dùng.

3. Phân quyền (Authorization):

- Dựa trên thông tin trong JWT (như roles hoặc quyền hạn), hệ thống sẽ quyết định quyền truy cập của người dùng vào các tài nguyên.
- Ví dụ, bệnh nhân chỉ có thể truy cập các thông tin cá nhân và lịch sử khám bệnh, trong khi bác sĩ có quyền truy cập thông tin hồ sơ bệnh án và tạo đơn thuốc. Quản trị viên sẽ có quyền quản lý toàn bộ hệ thống.
- Quá trình phân quyền này sẽ được xử lý ở mỗi service, dựa trên dữ liệu trong JWT và các điều kiện phân quyền cụ thể.

4. Làm mới token (Token Refresh):

- Để đảm bảo người dùng không phải đăng nhập lại quá thường xuyên, một cơ chế **refresh token** sẽ được triển khai.
- Khi JWT hết hạn, người dùng có thể sử dụng refresh token (một token dài hạn hơn) để yêu cầu một JWT mới mà không cần phải đăng nhập lại.

5. Đảm bảo an toàn cho JWT:

- **HTTPS:** Tất cả các giao tiếp giữa client và API đều phải được mã hóa qua giao thức **HTTPS** để đảm bảo an toàn khi truyền tải JWT.
- **Lưu trữ an toàn:** JWT sẽ được lưu trữ trong **localStorage** hoặc **sessionStorage** của trình duyệt ở phía client, tránh lưu trữ trong **cookie** (do dễ bị tấn công Cross-Site Scripting - XSS).
- **Chữ ký:** JWT được ký bằng một **secret key** hoặc **private/public key pair** (HMAC hoặc RSA) để đảm bảo rằng token không bị thay đổi trong quá trình truyền tải.

6. Xử lý Logout:

- Khi người dùng đăng xuất, hệ thống sẽ hủy bỏ JWT hiện tại và không cho phép sử dụng token đó nữa. Hệ thống sẽ xóa token khỏi phía client và không chấp nhận các yêu cầu với token đã hết hạn hoặc không hợp lệ.

2.1.7 Deployment Plan (Kế hoạch triển khai)

Tổng quan:

Hệ thống **E-commerce Health Care** sẽ được triển khai trên môi trường local bằng cách sử dụng **Docker** và **Docker Compose**. Mỗi service trong hệ thống sẽ được đóng gói trong một container riêng biệt, cùng với các cơ sở dữ liệu tương ứng. Docker Compose sẽ được sử dụng để quản lý và kết nối các container này một cách dễ dàng trong một mạng nội bộ. Điều này giúp đơn giản hóa quá trình triển khai và phát triển, đảm bảo tính nhất quán khi triển khai các dịch vụ khác nhau trong hệ thống.

Các bước triển khai:

1. Dockerize các Service:

- Mỗi service sẽ được đóng gói vào một Docker container riêng biệt. Các dịch vụ này bao gồm các API backend được xây dựng bằng **Django**, các frontend được xây dựng bằng **Angular**, và các cơ sở dữ liệu như **PostgreSQL**, **MySQL**, và **MongoDB**.
- Mỗi Docker container sẽ bao gồm tất cả các thành phần cần thiết cho việc chạy ứng dụng, bao gồm:
 - **Django app:** Dockerfile sẽ cài đặt các phụ thuộc cần thiết cho Django, thiết lập môi trường và chạy server Django.
 - **Angular frontend:** Dockerfile sẽ cài đặt các công cụ cần thiết để xây dựng và chạy ứng dụng Angular, đồng thời phục vụ các file tĩnh (static files) cho người dùng.

- **Cơ sở dữ liệu:** Các cơ sở dữ liệu PostgreSQL, MySQL, MongoDB sẽ được cấu hình trong các container riêng biệt, và mỗi service sẽ kết nối với cơ sở dữ liệu của riêng mình.

2. Sử dụng Docker Compose:

- **Docker Compose** sẽ được sử dụng để quản lý tất cả các container dịch vụ trong hệ thống. Nó sẽ giúp định nghĩa các container, mạng nội bộ và volume để lưu trữ dữ liệu.
- **docker-compose.yml** sẽ được cấu hình để khởi động tất cả các service và database. Các thông tin cấu hình cơ bản trong docker-compose.yml bao gồm:
 - **Frontend (Angular):** Chạy trên một container riêng, có thể tiếp cận API backend thông qua API Gateway.
 - **API Gateway (Nginx):** Container này sẽ nhận và chuyển tiếp các yêu cầu đến các service backend tương ứng, dựa trên các route API đã định nghĩa.
 - **Backend Services (Django):** Mỗi backend service (ví dụ: bệnh nhân, bác sĩ, quản trị viên) sẽ chạy trên một container riêng biệt với cấu hình Django và kết nối đến database tương ứng (PostgreSQL, MySQL, MongoDB).
 - **Databases (PostgreSQL, MySQL, MongoDB):** Mỗi service sẽ có database riêng biệt, không chia sẻ dữ liệu giữa các service.
 - **Redis:** Cung cấp bộ nhớ cache cho các dịch vụ cần tối ưu hóa hiệu suất.

3. Cấu hình Nginx làm API Gateway:

- **API Gateway** sẽ được triển khai dưới dạng một container riêng biệt, sử dụng **Nginx** làm reverse proxy để định tuyến các yêu cầu đến các backend services. Nginx sẽ giúp phân phối tải và bảo mật hệ thống bằng cách chỉ chấp nhận các yêu cầu hợp lệ với JWT.

4. Kết nối các Service qua Docker Network:

- Tất cả các dịch vụ (frontend, API Gateway, backend services, và databases) sẽ được kết nối với nhau thông qua một **Docker network**. Điều này đảm bảo rằng các service có thể giao tiếp với nhau mà không gặp phải vấn đề về kết nối mạng.
- Docker Compose sẽ tự động tạo ra mạng riêng cho các container này và cấu hình các container để chúng có thể giao tiếp với nhau qua tên dịch vụ.

5. Database Initialization:

- Các dịch vụ cơ sở dữ liệu (PostgreSQL, MySQL, MongoDB) sẽ được cấu hình để khởi tạo cơ sở dữ liệu và các bảng trong khi khởi động container lần đầu tiên.
- Các dữ liệu mẫu có thể được đưa vào trong quá trình khởi tạo để phục vụ cho quá trình kiểm thử và phát triển.

6. Environment Variables:

- Các biến môi trường sẽ được sử dụng để cấu hình các thông tin nhạy cảm như **database credentials**, **JWT secret**, **API keys** và các cấu hình khác. Các biến này sẽ được khai báo trong file .env và Docker Compose sẽ tự động đọc từ đó để cấu hình môi trường cho các container.

7. Quy trình triển khai:

- **Bước 1:** Tạo file docker-compose.yml để định nghĩa các service, container và network.
- **Bước 2:** Viết Dockerfile cho từng service (Django app, Angular frontend, Redis, Nginx, etc.).
- **Bước 3:** Cấu hình Nginx làm API Gateway, định tuyến các yêu cầu đến các service backend tương ứng.
- **Bước 4:** Sử dụng docker-compose up để xây dựng và khởi động tất cả các container.
- **Bước 5:** Kiểm tra và xác nhận tất cả các service đều hoạt động bình thường và có thể giao tiếp với nhau.

Môi trường triển khai:

- **Docker:** Cung cấp môi trường container cho tất cả các dịch vụ trong hệ thống.
- **Docker Compose:** Quản lý tất cả các dịch vụ và container liên quan trong một mạng nội bộ.
- **Nginx:** Làm API Gateway để điều hướng và bảo mật các yêu cầu.
- **Cơ sở dữ liệu:** PostgreSQL, MySQL, MongoDB được cấu hình trong các container riêng biệt cho từng service.
- **Redis:** Được sử dụng như một bộ nhớ cache cho các dịch vụ cần tối ưu hóa hiệu suất.

2.2 Design classes and methods in component

2.2.1 User Service

1. Overview

user-service chịu trách nhiệm quản lý thông tin định danh của người dùng toàn hệ thống, bao gồm: bệnh nhân, bác sĩ, y tá, dược sĩ, kỹ thuật viên, và quản trị viên. Đây là nơi xử lý xác thực (authentication), phân quyền (authorization), và cập nhật thông tin hồ sơ cơ bản.

2. Model Classes

2.1 User

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho người dùng
username	String (unique)	Tên đăng nhập hoặc số điện thoại/email
email	String (unique)	Email của người dùng
password	String (hashed)	Mật khẩu đã được mã hóa
role	Enum	Vai trò người dùng: PATIENT, DOCTOR, NURSE, PHARMACIST, TECHNICIAN, ADMIN
is_active	Boolean	Trạng thái hoạt động của tài khoản
created_at	DateTime	Ngày tạo
updated_at	DateTime	Ngày cập nhật gần nhất

2.2 UserProfile

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh hồ sơ người dùng

Trường	Kiểu dữ liệu	Mô tả
user_id	FK -> User	Khóa ngoại tới User
full_name	String	Họ và tên
gender	Enum	Giới tính
birth_date	Date	Ngày sinh
phone_number	String	Số điện thoại
address	String	Địa chỉ
insurance_number	String	Mã số bảo hiểm y tế
emergency_contact	String	Người liên hệ khẩn cấp

3. Business Methods

User

- `create_user(username, email, password, role)`
→ Tạo người dùng mới, hash password trước khi lưu.
- `check_password(raw_password)`
→ So sánh mật khẩu plaintext với hash.
- `deactivate()`
→ Vô hiệu hóa tài khoản người dùng.
- `activate()`
→ Kích hoạt lại tài khoản.

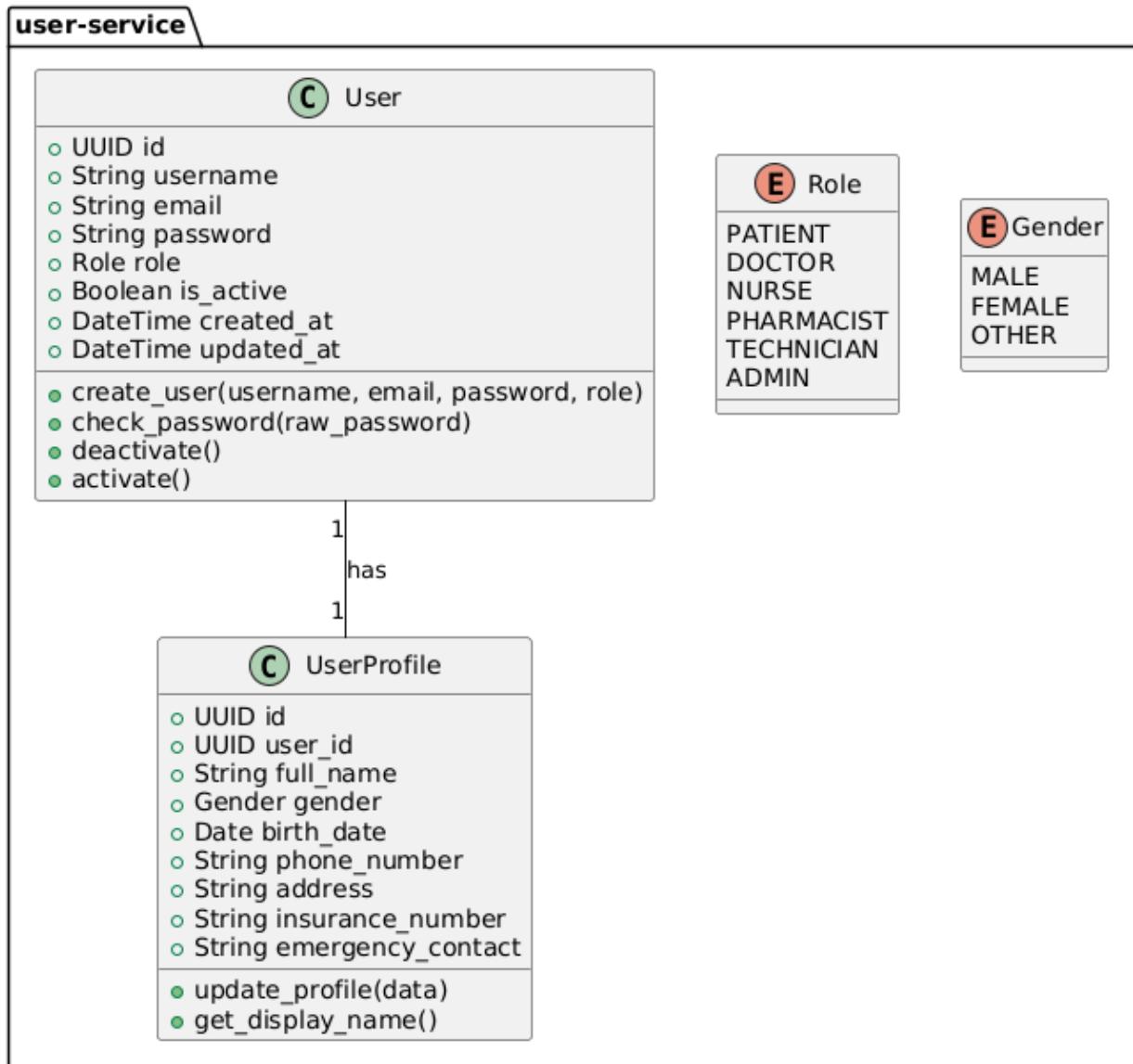
UserProfile

- `update_profile(data: dict)`
→ Cập nhật thông tin cá nhân.
 - `get_display_name()`
→ Trả về tên hiển thị (tùy theo cài đặt).
-

4. Relationships

- User **1 - 1** UserProfile (mỗi người dùng chỉ có một hồ sơ cá nhân)
- User được sử dụng bởi các service khác để phân quyền

5. UML



2.2.2 Appointment Service

1. Overview

appointment-service chịu trách nhiệm quản lý các lịch hẹn khám của người dùng (bệnh nhân và bác sĩ), bao gồm việc đặt lịch, hủy lịch, thay đổi lịch hẹn và theo dõi trạng thái của các lịch hẹn.

Dịch vụ này tương tác với user-service để xác thực thông tin bệnh nhân và bác sĩ, đồng thời quản lý trạng thái lịch hẹn qua các phương thức nghiệp vụ.

2. Model Classes

2.1 Appointment

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho lịch hẹn
patient_id	FK -> User	Khóa ngoại đến người bệnh
doctor_id	FK -> User	Khóa ngoại đến bác sĩ
appointment_time	DateTime	Thời gian khám
status	Enum	Trạng thái lịch hẹn: PENDING, CONFIRMED, COMPLETED, CANCELLED
created_at	DateTime	Ngày tạo lịch hẹn
updated_at	DateTime	Ngày cập nhật lịch hẹn

2.2 AppointmentHistory

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho lịch sử lịch hẹn
appointment_id	FK -> Appointment	Khóa ngoại đến lịch hẹn
change_type	Enum	Loại thay đổi: SCHEDULED, RESCHEDULED, CANCELLED
changed_at	DateTime	Thời gian thay đổi
note	String	Ghi chú thay đổi (nếu có)

3. Business Methods

Appointment

- `create_appointment(patient_id, doctor_id, appointment_time)`
→ Tạo mới lịch hẹn cho bệnh nhân và bác sĩ tại thời gian chỉ định. Lịch hẹn sẽ có trạng thái PENDING.
- `update_appointment_time(appointment_id, new_appointment_time)`
→ Thay đổi thời gian của một lịch hẹn đã có. Trạng thái lịch hẹn sẽ được cập nhật thành RESCHEDULED.
- `cancel_appointment(appointment_id)`
→ Hủy lịch hẹn và cập nhật trạng thái lịch hẹn thành CANCELLED.
- `confirm_appointment(appointment_id)`
→ Xác nhận lịch hẹn, cập nhật trạng thái thành CONFIRMED.
- `get_patient_appointments(patient_id)`
→ Lấy danh sách các lịch hẹn của bệnh nhân cụ thể.
- `get_doctor_appointments(doctor_id)`
→ Lấy danh sách các lịch hẹn của bác sĩ cụ thể.

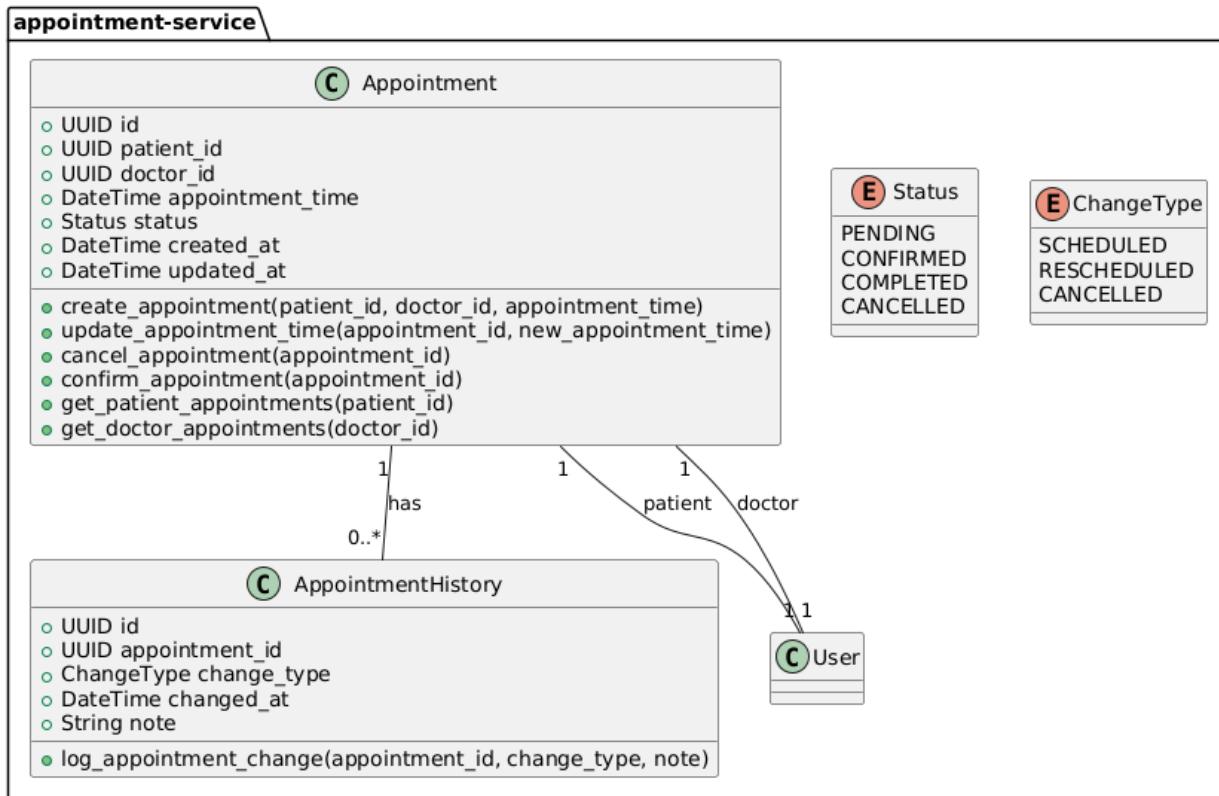
AppointmentHistory

- `log_appointment_change(appointment_id, change_type, note)`
→ Ghi lại lịch sử thay đổi của một lịch hẹn, lưu lại các thay đổi như reschedule, cancel...
-

4. Relationships

- **Appointment:** Liên kết với **User** (bệnh nhân và bác sĩ) qua khóa ngoại.
 - **AppointmentHistory:** Liên kết với **Appointment** để theo dõi các thay đổi của lịch hẹn.
-

5. UML Class Diagram



2.2.3 Medical Record Service

1. Overview

medical-record-service chịu trách nhiệm quản lý hồ sơ bệnh án của bệnh nhân. Dịch vụ này sẽ lưu trữ và cung cấp thông tin liên quan đến hồ sơ khám bệnh, bao gồm kết quả xét nghiệm, lịch sử bệnh án, đơn thuốc và các ghi chú của bác sĩ. Vì dữ liệu trong medical-record-service sẽ có tính linh hoạt cao và cần phải truy vấn theo nhiều điều kiện khác nhau, MongoDB sẽ là lựa chọn phù hợp, vì MongoDB hỗ trợ tốt các loại dữ liệu phi cấu trúc và truy vấn nhanh chóng.

2. Model Classes

2.1 MedicalRecord

Trường	Kiểu dữ liệu	Mô tả
id	ObjectId (MongoDB)	Định danh duy nhất cho hồ sơ bệnh án
patient_id	UUID	Khóa ngoại đến người bệnh (User)

Trường	Kiểu dữ liệu	Mô tả
doctor_id	UUID	Khóa ngoại đến bác sĩ (User)
appointment_id	UUID	Khóa ngoại đến lịch hẹn (Appointment)
diagnosis	String	Chẩn đoán bệnh của bác sĩ
prescription	Array of String	Danh sách các thuốc và chỉ định điều trị
test_results	Array of Object	Kết quả xét nghiệm (có thể bao gồm hình ảnh, PDF, kết quả văn bản)
notes	String	Ghi chú từ bác sĩ
created_at	DateTime	Ngày tạo hồ sơ bệnh án
updated_at	DateTime	Ngày cập nhật hồ sơ bệnh án

2.2 TestResult

Trường	Kiểu dữ liệu	Mô tả
id	ObjectId (MongoDB)	Định danh duy nhất cho kết quả xét nghiệm
medical_record_id	FK -> MedicalRecord	Khóa ngoại đến Medical Record
test_type	String	Loại xét nghiệm (máu, nước tiểu, X-quang, v.v.)
test_date	DateTime	Ngày thực hiện xét nghiệm
result	String	Kết quả xét nghiệm (có thể là văn bản hoặc hình ảnh đính kèm)
image	String	Đường dẫn tới hình ảnh kết quả xét nghiệm (nếu có)

3. Business Methods

MedicalRecord

- `create_medical_record(patient_id, doctor_id, appointment_id, diagnosis, prescription, test_results, notes)`
→ Tạo mới hồ sơ bệnh án cho bệnh nhân sau khi bác sĩ hoàn tất chẩn đoán và chỉ định điều trị.
- `update_medical_record(medical_record_id, diagnosis, prescription, test_results, notes)`
→ Cập nhật hồ sơ bệnh án khi có thay đổi về chẩn đoán, chỉ định điều trị hoặc kết quả xét nghiệm.
- `get_medical_record(patient_id)`
→ Lấy thông tin hồ sơ bệnh án của bệnh nhân, bao gồm tất cả các lần khám và các kết quả xét nghiệm đã thực hiện.
- `get_medical_records_by_doctor(doctor_id)`
→ Lấy thông tin các hồ sơ bệnh án mà bác sĩ đã khám cho bệnh nhân.
- `add_test_result(medical_record_id, test_type, test_date, result, image)`
→ Thêm kết quả xét nghiệm vào hồ sơ bệnh án, bao gồm kết quả văn bản và hình ảnh (nếu có).

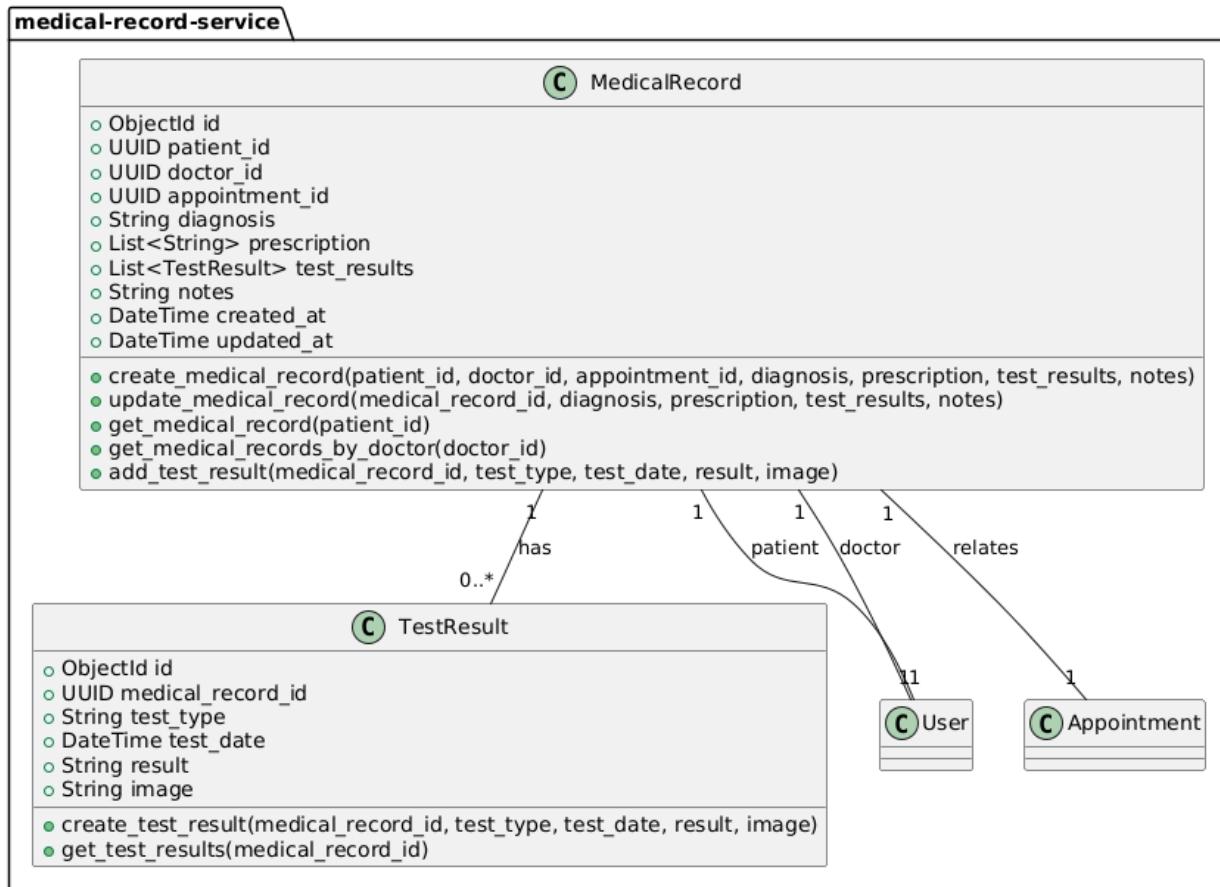
TestResult

- `create_test_result(medical_record_id, test_type, test_date, result, image)`
→ Tạo mới một kết quả xét nghiệm cho bệnh nhân.
 - `get_test_results(medical_record_id)`
→ Lấy tất cả các kết quả xét nghiệm đã được lưu trữ cho hồ sơ bệnh án.
-

4. Relationships

- **MedicalRecord**: Liên kết với **User** (bệnh nhân và bác sĩ) qua khóa ngoại.
 - **MedicalRecord**: Liên kết với **Appointment** để theo dõi lịch sử khám của bệnh nhân.
 - **TestResult**: Liên kết với **MedicalRecord** để lưu trữ các kết quả xét nghiệm của bệnh nhân.
-

5. UML Class Diagram



2.2.4 Prescription Service

1. Overview

prescription-service chịu trách nhiệm quản lý các đơn thuốc của bác sĩ. Dịch vụ này sẽ cung cấp chức năng tạo, cập nhật, truy vấn và theo dõi các đơn thuốc của bệnh nhân, bao gồm thông tin thuốc, liều dùng, chỉ định của bác sĩ và các ghi chú liên quan. PostgreSQL được chọn là cơ sở dữ liệu cho dịch vụ này vì tính nhất quán, khả năng xử lý giao dịch mạnh mẽ, và khả năng hỗ trợ các truy vấn quan hệ phức tạp.

2. Model Classes

2.1 Prescription

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho đơn thuốc

Trường	Kiểu dữ liệu	Mô tả
patient_id	UUID	Khóa ngoại đến bảng người bệnh (User)
doctor_id	UUID	Khóa ngoại đến bác sĩ (User)
appointment_id	UUID	Khóa ngoại đến lịch hẹn khám (Appointment)
medicine_list	Array of JSON	Danh sách thuốc bao gồm tên thuốc, liều dùng và cách sử dụng
instructions	String	Chỉ định chi tiết từ bác sĩ
created_at	DateTime	Thời gian tạo đơn thuốc
updated_at	DateTime	Thời gian cập nhật đơn thuốc

2.2 Medicine

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho thuốc
name	String	Tên thuốc
dosage	String	Liều dùng của thuốc
instruction	String	Cách sử dụng thuốc
quantity	Integer	Số lượng thuốc trong đơn
prescription_id	FK -> Prescription	Khóa ngoại đến Prescription

3. Business Methods

Prescription

- `create_prescription(patient_id, doctor_id, appointment_id, medicine_list, instructions)`
→ Tạo đơn thuốc mới cho bệnh nhân, bao gồm thông tin thuốc và chỉ định từ bác sĩ.
- `update_prescription(prescription_id, medicine_list, instructions)`
→ Cập nhật đơn thuốc cho bệnh nhân nếu có thay đổi về thuốc, liều dùng, hoặc chỉ định.

- `get_prescriptions_by_patient(patient_id)`
→ Lấy tất cả đơn thuốc của bệnh nhân.
- `get_prescriptions_by_doctor(doctor_id)`
→ Lấy tất cả đơn thuốc của bác sĩ cho các bệnh nhân mình đã khám.
- `get_prescription_details(prescription_id)`
→ Lấy chi tiết đơn thuốc theo ID đơn thuốc.

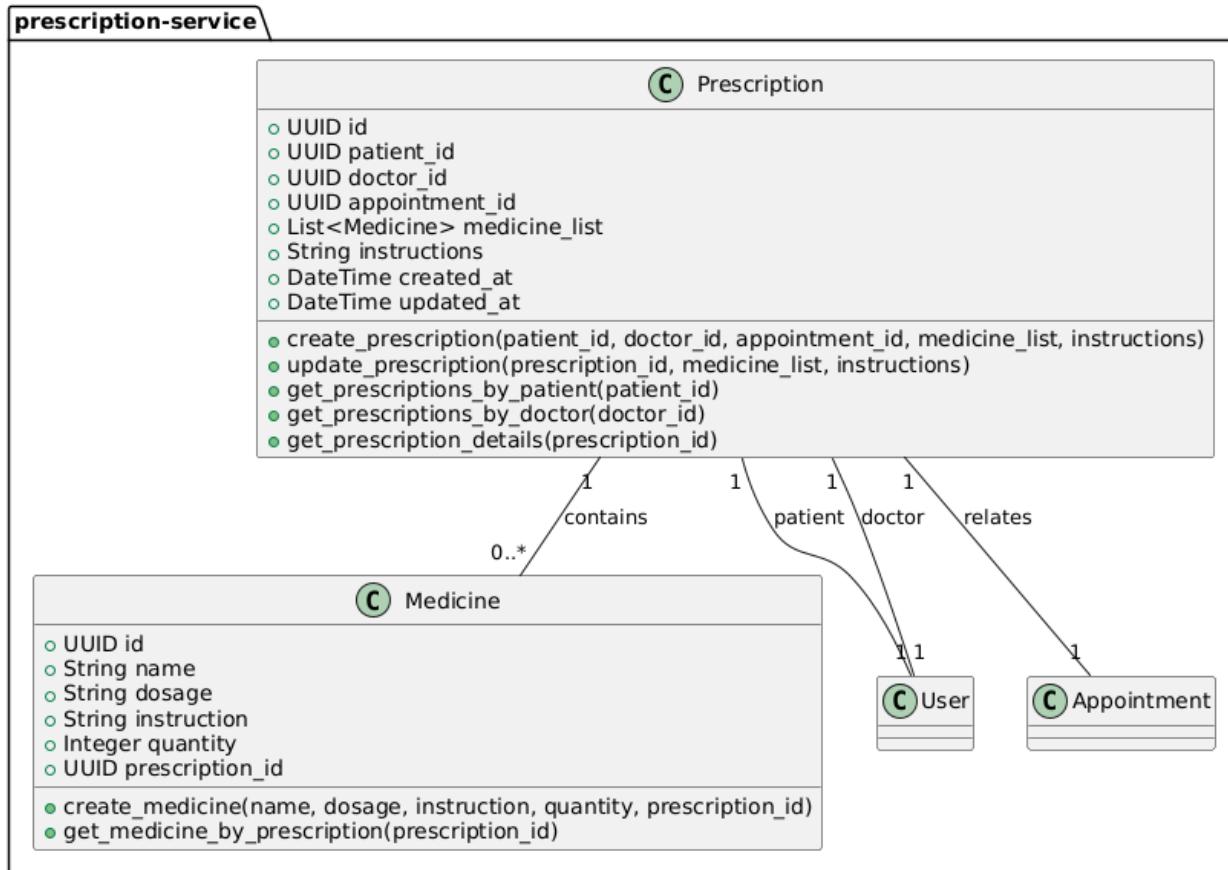
Medicine

- `create_medicine(name, dosage, instruction, quantity, prescription_id)`
→ Tạo một loại thuốc trong đơn thuốc.
 - `get_medicine_by_prescription(prescription_id)`
→ Lấy danh sách thuốc trong một đơn thuốc cụ thể.
-

4. Relationships

- **Prescription:** Liên kết với **User** (bệnh nhân và bác sĩ) qua khóa ngoại.
 - **Prescription:** Liên kết với **Appointment** để theo dõi đơn thuốc được tạo từ cuộc hẹn khám.
 - **Medicine:** Liên kết với **Prescription** để lưu trữ thông tin các thuốc trong đơn thuốc.
-

5. UML Class Diagram



2.2.5 Pharmacy Service

1. Overview

pharmacy-service chịu trách nhiệm quản lý thông tin dược phẩm trong hệ thống, bao gồm việc theo dõi tồn kho thuốc, cập nhật thông tin về các loại thuốc, số lượng, hạn sử dụng, và các tác vụ liên quan đến việc cấp phát thuốc cho bệnh nhân. Dịch vụ này sẽ sử dụng MySQL vì tính khả thi của việc quản lý dữ liệu có cấu trúc như thuốc, số lượng, và các giao dịch liên quan đến cấp phát thuốc.

2. Model Classes

2.1 Medicine

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho thuốc

Trường	Kiểu dữ liệu	Mô tả
name	String	Tên thuốc
description	String	Mô tả chi tiết về thuốc
price	Decimal	Giá thuốc
quantity_in_stock	Integer	Số lượng thuốc trong kho
expiry_date	Date	Ngày hết hạn của thuốc
created_at	DateTime	Thời gian thuốc được thêm vào hệ thống
updated_at	DateTime	Thời gian thuốc được cập nhật

2.2 Prescription_Medicine

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất của mỗi quan hệ thuốc và đơn thuốc
prescription_id	UUID	Khóa ngoại đến Prescription
medicine_id	UUID	Khóa ngoại đến Medicine
quantity	Integer	Số lượng thuốc trong đơn thuốc

3. Business Methods

Medicine

- `create_medicine(name, description, price, quantity_in_stock, expiry_date)`
→ Tạo mới thuốc trong kho dược phẩm, bao gồm tên thuốc, mô tả, giá cả, số lượng, và hạn sử dụng.
- `update_medicine(medicine_id, name, description, price, quantity_in_stock, expiry_date)`
→ Cập nhật thông tin thuốc trong kho.
- `get_medicine_by_id(medicine_id)`
→ Lấy thông tin chi tiết của thuốc theo ID.

- `get_all_medicines()`
→ Lấy danh sách tất cả các thuốc có trong kho.
- `decrement_quantity_in_stock(medicine_id, quantity)`
→ Giảm số lượng thuốc trong kho khi có đơn thuốc hoặc cấp phát.

Prescription_Medicine

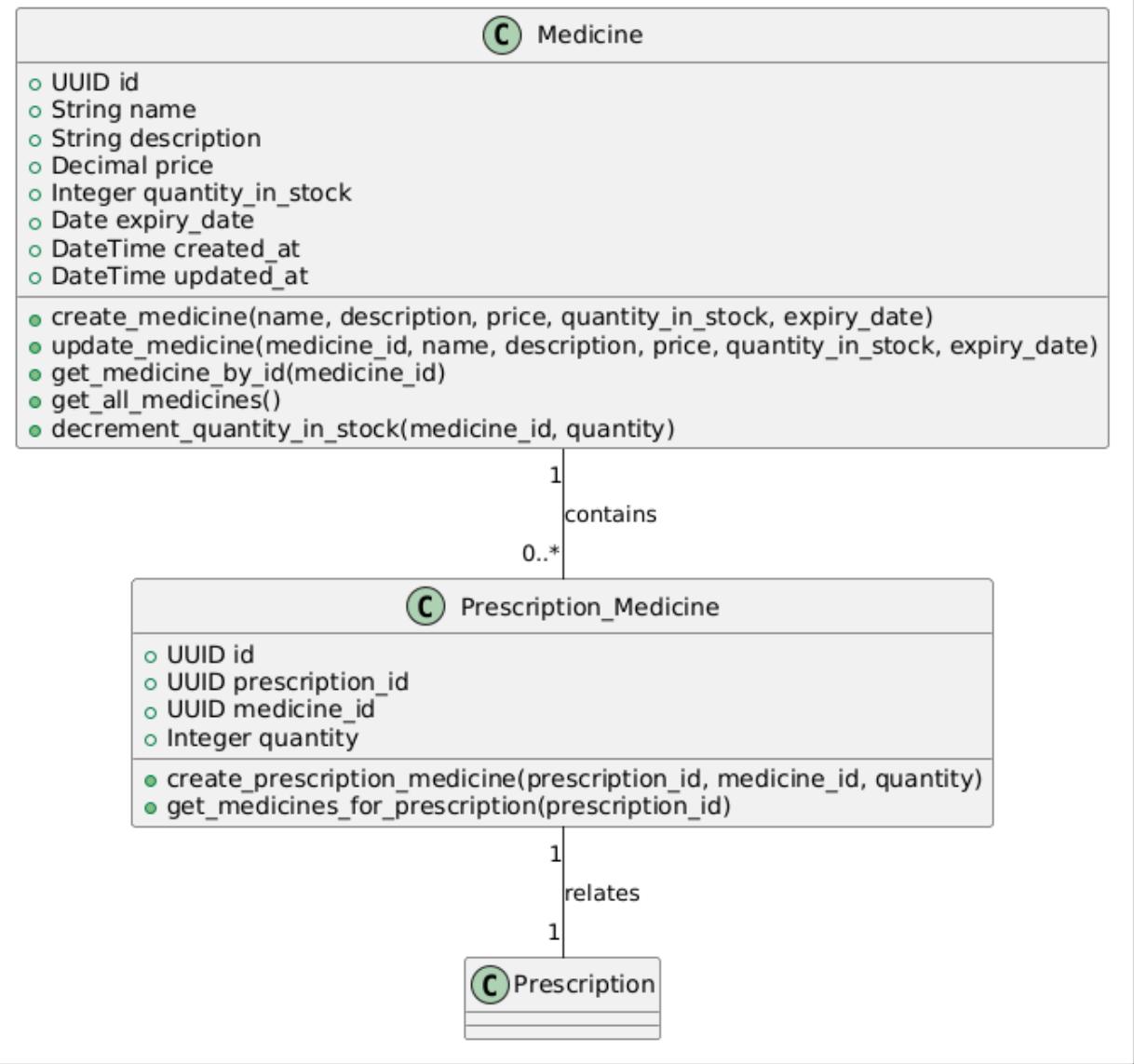
- `create_prescription_medicine(prescription_id, medicine_id, quantity)`
→ Tạo mối quan hệ giữa đơn thuốc và thuốc, lưu trữ thông tin số lượng thuốc trong đơn.
 - `get_medicines_for_prescription(prescription_id)`
→ Lấy danh sách thuốc liên quan đến đơn thuốc, bao gồm thông tin số lượng và thuốc.
-

4. Relationships

- **Medicine:** Có một mối quan hệ một-nhiều với **Prescription_Medicine** để lưu trữ các đơn thuốc có sử dụng thuốc.
 - **Prescription_Medicine:** Liên kết với **Prescription** để ghi lại thuốc nào được chỉ định trong mỗi đơn thuốc.
-

5. UML Class Diagram

pharmacy-service



2.2.6 Billing Service

1. Overview

billing-service chịu trách nhiệm quản lý thông tin hóa đơn và thanh toán trong hệ thống, bao gồm việc tạo hóa đơn cho các dịch vụ khám bệnh, xét nghiệm, thuốc, và các dịch vụ khác mà bệnh nhân đã sử dụng. Dịch vụ này cũng sẽ xử lý việc thanh toán và theo dõi tình trạng thanh toán của các hóa đơn. PostgreSQL được chọn để lưu trữ dữ liệu liên quan đến hóa đơn vì tính chất dữ liệu có cấu trúc, quan hệ chặt chẽ giữa các hóa đơn và dịch vụ.

2. Model Classes

2.1 Bill

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho hóa đơn
patient_id	UUID	Khóa ngoại đến Patient
amount	Decimal	Tổng số tiền của hóa đơn
status	Enum (Pending, Paid, Cancelled)	Trạng thái hóa đơn (chờ thanh toán, đã thanh toán, hủy)
created_at	DateTime	Thời gian tạo hóa đơn
updated_at	DateTime	Thời gian cập nhật hóa đơn
due_date	DateTime	Ngày hết hạn thanh toán

2.2 Payment

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho thanh toán
bill_id	UUID	Khóa ngoại đến Bill
payment_method	Enum (CreditCard, Cash, Insurance)	Phương thức thanh toán
amount	Decimal	Số tiền thanh toán
status	Enum (Completed, Pending, Failed)	Trạng thái thanh toán
payment_date	DateTime	Ngày thanh toán

3. Business Methods

Bill

- `create_bill(patient_id, amount, status, due_date)`
→ Tạo mới hóa đơn cho bệnh nhân, bao gồm tổng số tiền, trạng thái, và ngày hết hạn thanh toán.
- `update_bill(bill_id, amount, status)`
→ Cập nhật thông tin hóa đơn như số tiền, trạng thái thanh toán.
- `get_bill_by_id(bill_id)`
→ Lấy thông tin chi tiết của hóa đơn theo ID.
- `get_bills_by_patient(patient_id)`
→ Lấy danh sách các hóa đơn của một bệnh nhân.

Payment

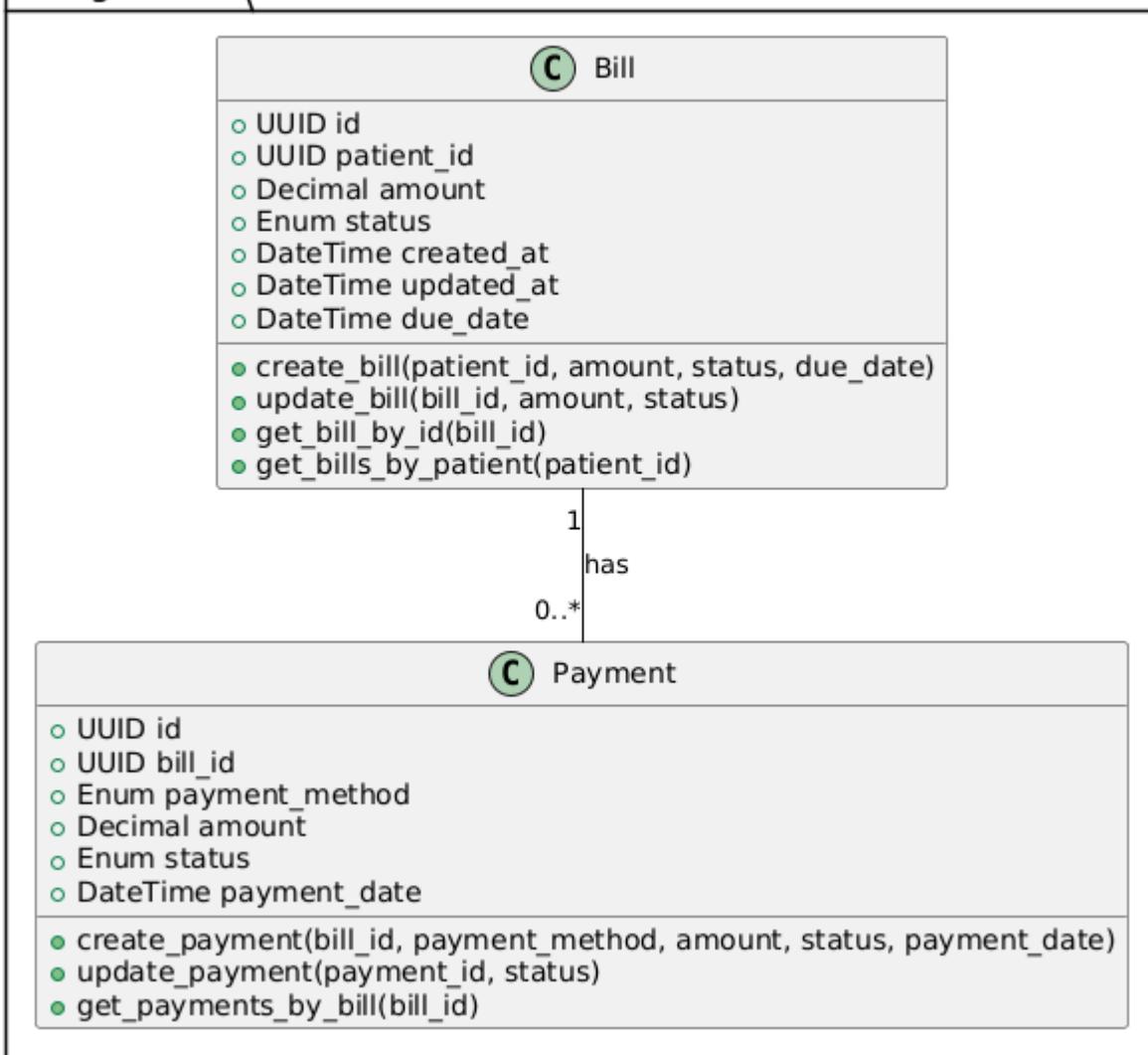
- `create_payment(bill_id, payment_method, amount, status, payment_date)`
→ Tạo mới thanh toán cho một hóa đơn, bao gồm phương thức thanh toán và số tiền thanh toán.
- `update_payment(payment_id, status)`
→ Cập nhật trạng thái thanh toán (hoàn thành, chờ, thất bại).
- `get_payments_by_bill(bill_id)`
→ Lấy danh sách các thanh toán liên quan đến một hóa đơn.

4. Relationships

- **Bill:** Một hóa đơn có thể có nhiều thanh toán (1-n với **Payment**).
 - **Payment:** Liên kết với một hóa đơn duy nhất, chỉ ra thanh toán của hóa đơn đó.
-

5. UML Class Diagram

billing-service



2.2.7 Notification Service

1. Overview

notification-service chịu trách nhiệm gửi thông báo cho người dùng trong hệ thống về các sự kiện quan trọng như xác nhận lịch khám, nhắc nhở tái khám, thông báo kết quả xét nghiệm, và các cập nhật liên quan đến tài khoản bệnh nhân, bác sĩ, dược sĩ, v.v. MongoDB được chọn để lưu trữ thông tin liên quan đến các thông báo vì tính linh hoạt của nó trong việc xử lý dữ liệu không đồng nhất và hỗ trợ mở rộng quy mô.

2. Model Classes

2.1 Notification

Trường	Kiểu dữ liệu	Mô tả
id	ObjectId	Định danh duy nhất cho thông báo
user_id	UUID	Khóa ngoại đến người dùng nhận thông báo (bệnh nhân, bác sĩ, dược sĩ, v.v.)
message	String	Nội dung thông báo
type	Enum (Appointment, Payment, LabResult, etc.)	Loại thông báo
status	Enum (Sent, Pending, Failed)	Trạng thái thông báo
created_at	DateTime	Thời gian tạo thông báo
updated_at	DateTime	Thời gian cập nhật thông báo
read	Boolean	Trạng thái đã đọc (true/false)

3. Business Methods

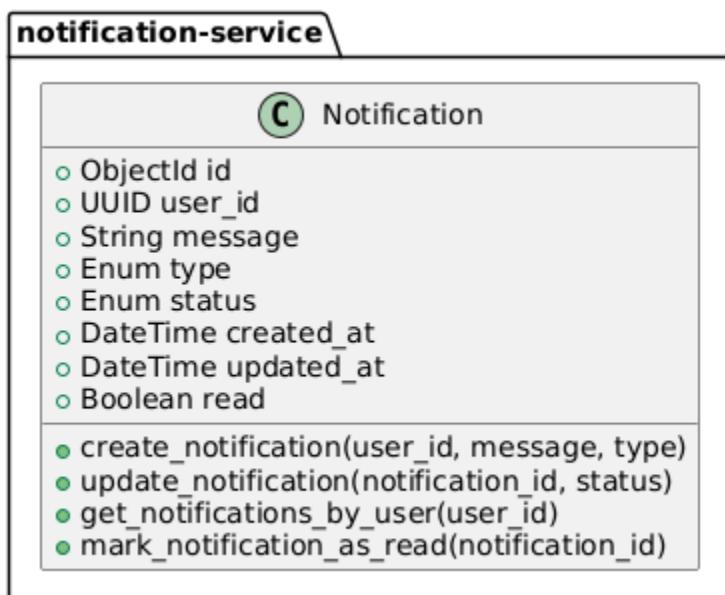
Notification

- `create_notification(user_id, message, type)`
→ Tạo mới một thông báo cho người dùng với nội dung và loại thông báo xác định.
- `update_notification(notification_id, status)`
→ Cập nhật trạng thái của thông báo (đã gửi, chờ, thất bại).
- `get_notifications_by_user(user_id)`
→ Lấy danh sách các thông báo của người dùng.
- `mark_notification_as_read(notification_id)`
→ Đánh dấu thông báo là đã đọc.

4. Relationships

- **Notification:** Mỗi thông báo chỉ thuộc về một người dùng, do đó, nó có quan hệ 1-n với người dùng (bệnh nhân, bác sĩ, v.v.). MongoDB cho phép dễ dàng lưu trữ các thông báo không đồng nhất (có thể có nhiều loại thông báo khác nhau).

5. UML Class Diagram



2.2.8 Insurance Service

1. Overview

insurance-service chịu trách nhiệm quản lý các yêu cầu bảo hiểm của bệnh nhân, bao gồm việc xử lý các yêu cầu bảo hiểm, theo dõi trạng thái yêu cầu, cập nhật thông tin bảo hiểm, và xác thực các yêu cầu bảo hiểm. PostgreSQL được chọn làm cơ sở dữ liệu cho service này vì tính toàn vẹn dữ liệu mạnh mẽ và hỗ trợ các truy vấn phức tạp, cần thiết cho việc quản lý các hồ sơ bảo hiểm.

2. Model Classes

2.1 InsuranceClaim

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho yêu cầu bảo hiểm
patient_id	UUID	Khóa ngoại đến bệnh nhân
claim_number	String	Mã số yêu cầu bảo hiểm

Trường	Kiểu dữ liệu	Mô tả
amount_claimed	Decimal	Số tiền yêu cầu bảo hiểm
status	Enum (Pending, Approved, Rejected)	Trạng thái yêu cầu bảo hiểm
date_of_claim	DateTime	Ngày yêu cầu bảo hiểm
updated_at	DateTime	Thời gian cập nhật yêu cầu bảo hiểm
created_at	DateTime	Thời gian tạo yêu cầu bảo hiểm

2.2 InsurancePolicy

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho chính sách bảo hiểm
policy_number	String	Số hợp đồng bảo hiểm
patient_id	UUID	Khóa ngoại đến bệnh nhân
coverage_amount	Decimal	Số tiền bảo hiểm được bảo vệ
policy_type	Enum (Health, Life, etc.)	Loại hợp đồng bảo hiểm
start_date	DateTime	Ngày bắt đầu hợp đồng bảo hiểm
end_date	DateTime	Ngày hết hạn hợp đồng bảo hiểm
status	Enum (Active, Expired, Suspended)	Trạng thái của hợp đồng bảo hiểm

3. Business Methods

InsuranceClaim

- `create_claim(patient_id, claim_number, amount_claimed, date_of_claim)`
→ Tạo mới một yêu cầu bảo hiểm với thông tin bệnh nhân và các thông tin yêu cầu bảo hiểm.

- `update_claim_status(claim_id, status)`
→ Cập nhật trạng thái của yêu cầu bảo hiểm (đang chờ, đã phê duyệt, bị từ chối).
- `get_claims_by_patient(patient_id)`
→ Lấy danh sách các yêu cầu bảo hiểm của bệnh nhân.
- `get_claim_by_claim_number(claim_number)`
→ Lấy thông tin yêu cầu bảo hiểm qua mã số yêu cầu bảo hiểm.

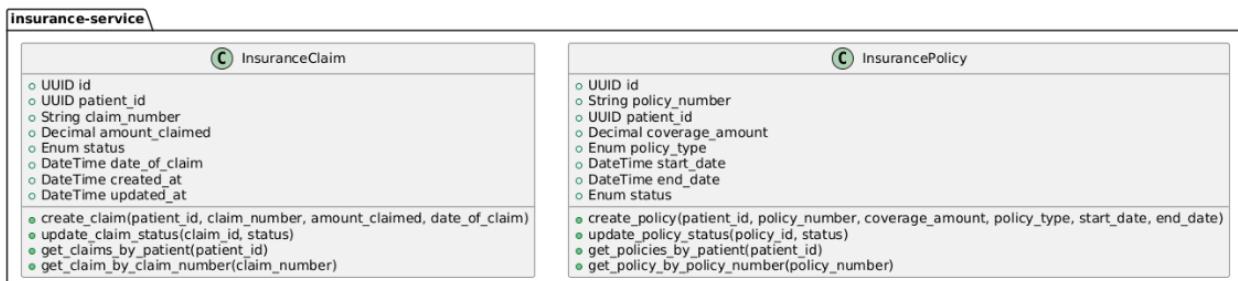
InsurancePolicy

- `create_policy(patient_id, policy_number, coverage_amount, policy_type, start_date, end_date)`
→ Tạo mới một chính sách bảo hiểm cho bệnh nhân.
- `update_policy_status(policy_id, status)`
→ Cập nhật trạng thái của chính sách bảo hiểm (đang hoạt động, đã hết hạn, tạm ngừng).
- `get_policies_by_patient(patient_id)`
→ Lấy danh sách các chính sách bảo hiểm của bệnh nhân.
- `get_policy_by_policy_number(policy_number)`
→ Lấy thông tin chính sách bảo hiểm qua số hợp đồng bảo hiểm.

4. Relationships

- **InsuranceClaim:** Mỗi yêu cầu bảo hiểm thuộc về một bệnh nhân, vì vậy có quan hệ 1-n với bệnh nhân.
- **InsurancePolicy:** Mỗi chính sách bảo hiểm thuộc về một bệnh nhân và có quan hệ 1-n với bệnh nhân. Một bệnh nhân có thể có nhiều chính sách bảo hiểm.

5. UML Class Diagram



2.2.9 Laboratory Service

1. Overview

laboratory-service chịu trách nhiệm quản lý các yêu cầu xét nghiệm của bệnh nhân, bao gồm việc nhận chỉ định xét nghiệm từ bác sĩ, thực hiện các xét nghiệm (máu, nước tiểu, siêu âm, X-quang,...) và lưu trữ kết quả xét nghiệm dưới dạng các file đính kèm (PDF, hình ảnh). MongoDB được chọn làm cơ sở dữ liệu cho service này vì tính linh hoạt và khả năng lưu trữ các tài liệu lớn và không đồng nhất, phù hợp với việc lưu trữ các kết quả xét nghiệm đa dạng dạng dữ liệu (hình ảnh, file PDF).

2. Model Classes

2.1 TestOrder

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho đơn yêu cầu xét nghiệm
patient_id	UUID	Khóa ngoại đến bệnh nhân
doctor_id	UUID	Khóa ngoại đến bác sĩ yêu cầu xét nghiệm
test_type	Enum (Blood, Urine, X-ray, Ultrasound, etc.)	Loại xét nghiệm
order_date	DateTime	Ngày tạo đơn yêu cầu xét nghiệm
status	Enum (Pending, In Progress, Completed)	Trạng thái của yêu cầu xét nghiệm
result	String	Kết quả xét nghiệm (nếu có)
result_file	String	Đường dẫn đến file kết quả xét nghiệm (PDF, ảnh)

2.2 TestResult

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho kết quả xét nghiệm

Trường	Kiểu dữ liệu	Mô tả
test_order_id	UUID	Khóa ngoại đến đơn yêu cầu xét nghiệm
result_data	JSON	Dữ liệu kết quả xét nghiệm (có thể là hình ảnh, giá trị số, văn bản, tùy thuộc vào loại xét nghiệm)
result_file	String	Đường dẫn đến file kết quả (nếu có)
date_of_result	DateTime	Ngày có kết quả xét nghiệm
status	Enum (Pending, Completed, Rejected)	Trạng thái của kết quả xét nghiệm

3. Business Methods

TestOrder

- `create_test_order(patient_id, doctor_id, test_type, order_date)`
→ Tạo mới một đơn yêu cầu xét nghiệm, bao gồm các thông tin bệnh nhân và bác sĩ yêu cầu.
- `update_test_order_status(order_id, status)`
→ Cập nhật trạng thái của đơn yêu cầu xét nghiệm (chờ xử lý, đang tiến hành, hoàn thành).
- `get_test_orders_by_patient(patient_id)`
→ Lấy danh sách các đơn yêu cầu xét nghiệm của bệnh nhân.
- `get_test_order_by_order_id(order_id)`
→ Lấy thông tin chi tiết của một đơn yêu cầu xét nghiệm qua mã đơn.

TestResult

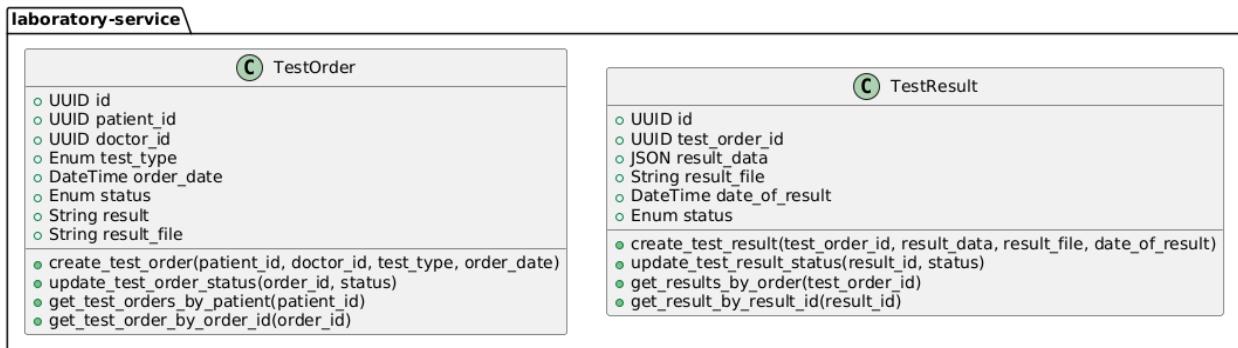
- `create_test_result(test_order_id, result_data, result_file, date_of_result)`
→ Tạo kết quả xét nghiệm cho một đơn yêu cầu xét nghiệm.
- `update_test_result_status(result_id, status)`
→ Cập nhật trạng thái của kết quả xét nghiệm (chờ xử lý, đã hoàn thành, bị từ chối).
- `get_results_by_order(test_order_id)`
→ Lấy kết quả xét nghiệm từ đơn yêu cầu xét nghiệm.

- `get_result_by_result_id(result_id)`
→ Lấy thông tin chi tiết của kết quả xét nghiệm qua mã kết quả.
-

4. Relationships

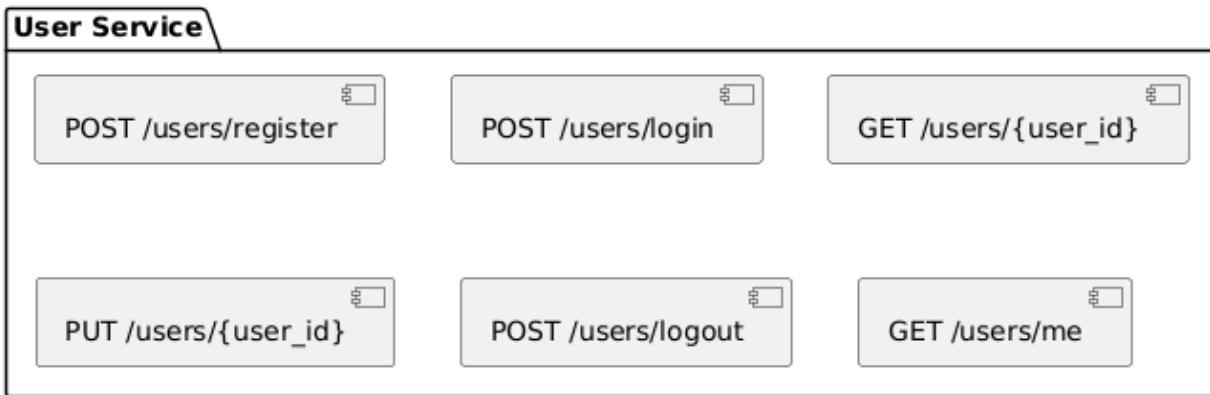
- **TestOrder:** Mỗi đơn yêu cầu xét nghiệm thuộc về một bệnh nhân và bác sĩ. Một bệnh nhân có thể có nhiều đơn yêu cầu xét nghiệm.
 - **TestResult:** Mỗi kết quả xét nghiệm liên kết với một đơn yêu cầu xét nghiệm và có thể có nhiều kết quả cho một đơn.
-

5. UML Class Diagram



2.3 Design API

2.3.1 User service



1. POST /users/register

- **Mô tả:** Đăng ký một người dùng mới vào hệ thống.
- **Tham số yêu cầu:**
 - username (String): Tên người dùng (Unique).
 - email (String): Địa chỉ email của người dùng.
 - password (String): Mật khẩu người dùng.
 - full_name (String): Tên đầy đủ của người dùng.
- **Mã trạng thái:**
 - 201 Created: Thành công, người dùng đã được đăng ký.
 - 400 Bad Request: Thiếu thông tin hoặc email đã tồn tại.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. POST /users/login

- **Mô tả:** Đăng nhập người dùng vào hệ thống.
- **Tham số yêu cầu:**
 - email (String): Địa chỉ email của người dùng.
 - password (String): Mật khẩu của người dùng.
- **Mã trạng thái:**

- 200 OK: Thành công, trả về token JWT.
- 401 Unauthorized: Sai email hoặc mật khẩu.
- 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /users/{user_id}

- **Mô tả:** Lấy thông tin chi tiết của người dùng bằng user_id.
- **Tham số yêu cầu:**
 - user_id (String): ID của người dùng cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin người dùng.
 - 404 Not Found: Không tìm thấy người dùng với user_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

4. PUT /users/{user_id}

- **Mô tả:** Cập nhật thông tin người dùng.
- **Tham số yêu cầu:**
 - user_id (String): ID của người dùng cần cập nhật.
 - Dữ liệu yêu cầu: email, full_name, phone_number, và các trường khác có thể cập nhật.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin người dùng đã cập nhật.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 404 Not Found: Không tìm thấy người dùng với user_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. POST /users/logout

- **Mô tả:** Đăng xuất người dùng, xóa token JWT.
- **Tham số yêu cầu:**
 - token (String): Token JWT cần xóa.

- **Mã trạng thái:**
 - 200 OK: Thành công, người dùng đã đăng xuất.
 - 400 Bad Request: Token không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

6. GET /users/me

- **Mô tả:** Lấy thông tin người dùng đã đăng nhập (dựa trên token).
- **Tham số yêu cầu:**
 - Authorization (String): Bearer token.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin người dùng.
 - 401 Unauthorized: Token không hợp lệ hoặc hết hạn.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.2 Appointment Service



1. POST /appointments/create

- **Mô tả:** Tạo một cuộc hẹn mới giữa bệnh nhân và bác sĩ.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân.
 - doctor_id (String): ID của bác sĩ.
 - appointment_date (String): Ngày giờ của cuộc hẹn.
 - appointment_type (String): Loại hẹn (ví dụ: "offline", "online").
 - reason (String): Lý do hẹn khám.
- **Mã trạng thái:**

- 201 Created: Cuộc hẹn đã được tạo thành công.
- 400 Bad Request: Dữ liệu không hợp lệ (ví dụ: ngày giờ không hợp lệ).
- 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /appointments/{appointment_id}

- **Mô tả:** Lấy thông tin chi tiết của một cuộc hẹn theo appointment_id.
- **Tham số yêu cầu:**
 - appointment_id (String): ID của cuộc hẹn cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin cuộc hẹn.
 - 404 Not Found: Không tìm thấy cuộc hẹn với appointment_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /appointments/patient/{patient_id}

- **Mô tả:** Lấy tất cả cuộc hẹn của bệnh nhân theo patient_id.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân cần xem các cuộc hẹn.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các cuộc hẹn của bệnh nhân.
 - 404 Not Found: Không có cuộc hẹn nào cho bệnh nhân.
 - 500 Internal Server Error: Lỗi từ máy chủ.

4. GET /appointments/doctor/{doctor_id}

- **Mô tả:** Lấy tất cả cuộc hẹn của bác sĩ theo doctor_id.
- **Tham số yêu cầu:**
 - doctor_id (String): ID của bác sĩ cần xem các cuộc hẹn.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các cuộc hẹn của bác sĩ.
 - 404 Not Found: Không có cuộc hẹn nào cho bác sĩ.

- 500 Internal Server Error: Lỗi từ máy chủ.

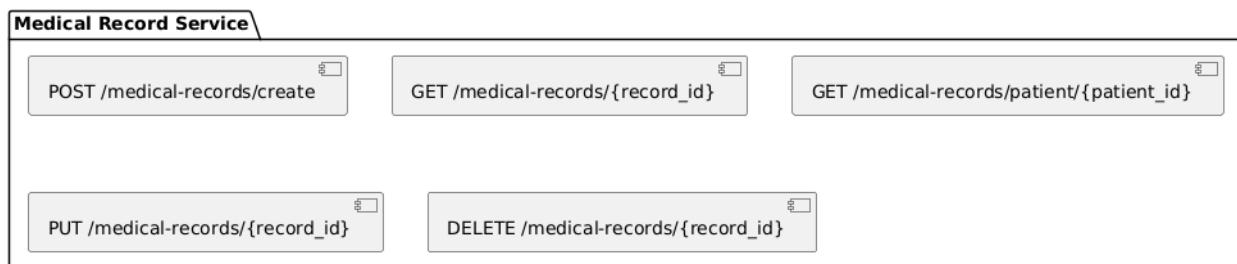
5. PUT /appointments/{appointment_id}

- **Mô tả:** Cập nhật thông tin cuộc hẹn theo appointment_id.
- **Tham số yêu cầu:**
 - appointment_id (String): ID của cuộc hẹn cần cập nhật.
 - Dữ liệu yêu cầu: appointment_date, appointment_type, reason (các trường có thể thay đổi).
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin cuộc hẹn đã cập nhật.
 - 404 Not Found: Không tìm thấy cuộc hẹn với appointment_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

6. DELETE /appointments/{appointment_id}

- **Mô tả:** Hủy cuộc hẹn theo appointment_id.
- **Tham số yêu cầu:**
 - appointment_id (String): ID của cuộc hẹn cần hủy.
- **Mã trạng thái:**
 - 200 OK: Thành công, cuộc hẹn đã bị hủy.
 - 404 Not Found: Không tìm thấy cuộc hẹn với appointment_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.3 Medical Record Service API Design



1. POST /medical-records/create

- **Mô tả:** Tạo hồ sơ y tế cho bệnh nhân mới.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân.
 - record_type (String): Loại hồ sơ (ví dụ: "general", "surgery").
 - record_details (String): Chi tiết hồ sơ y tế.
 - record_date (String): Ngày của hồ sơ y tế.
- **Mã trạng thái:**
 - 201 Created: Hồ sơ y tế đã được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ (ví dụ: thiếu patient_id).
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /medical-records/{record_id}

- **Mô tả:** Lấy thông tin chi tiết của một hồ sơ y tế theo record_id.
- **Tham số yêu cầu:**
 - record_id (String): ID của hồ sơ y tế cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin hồ sơ y tế.
 - 404 Not Found: Không tìm thấy hồ sơ y tế với record_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /medical-records/patient/{patient_id}

- **Mô tả:** Lấy tất cả hồ sơ y tế của bệnh nhân theo patient_id.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân cần lấy hồ sơ.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các hồ sơ y tế của bệnh nhân.
 - 404 Not Found: Không có hồ sơ nào cho bệnh nhân.
 - 500 Internal Server Error: Lỗi từ máy chủ.

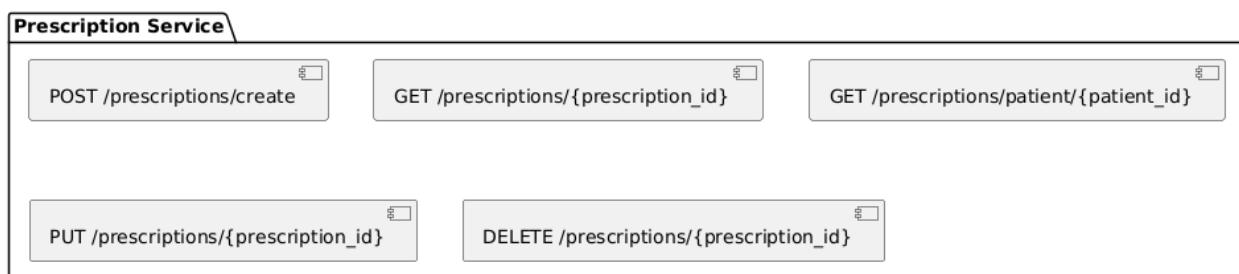
4. PUT /medical-records/{record_id}

- **Mô tả:** Cập nhật thông tin hồ sơ y tế theo record_id.
- **Tham số yêu cầu:**
 - record_id (String): ID của hồ sơ cần cập nhật.
 - record_type (String): Loại hồ sơ y tế (có thể thay đổi).
 - record_details (String): Chi tiết mới của hồ sơ.
 - record_date (String): Ngày của hồ sơ (có thể thay đổi).
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin hồ sơ đã cập nhật.
 - 404 Not Found: Không tìm thấy hồ sơ y tế với record_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /medical-records/{record_id}

- **Mô tả:** Xóa hồ sơ y tế theo record_id.
- **Tham số yêu cầu:**
 - record_id (String): ID của hồ sơ cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, hồ sơ y tế đã bị xóa.
 - 404 Not Found: Không tìm thấy hồ sơ y tế với record_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.4 Prescription Service API Design



1. POST /prescriptions/create

- **Mô tả:** Tạo đơn thuốc cho bệnh nhân.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân.
 - doctor_id (String): ID của bác sĩ.
 - medications (Array): Danh sách các thuốc và liều lượng.
 - prescription_date (String): Ngày cấp đơn thuốc.
- **Mã trạng thái:**
 - 201 Created: Đơn thuốc đã được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ (ví dụ: thiếu patient_id hoặc medications).
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /prescriptions/{prescription_id}

- **Mô tả:** Lấy thông tin chi tiết của một đơn thuốc theo prescription_id.
- **Tham số yêu cầu:**
 - prescription_id (String): ID của đơn thuốc cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin đơn thuốc.
 - 404 Not Found: Không tìm thấy đơn thuốc với prescription_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /prescriptions/patient/{patient_id}

- **Mô tả:** Lấy tất cả đơn thuốc của một bệnh nhân theo patient_id.
- **Tham số yêu cầu:**
 - patient_id (String): ID của bệnh nhân cần lấy danh sách đơn thuốc.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các đơn thuốc của bệnh nhân.

- 404 Not Found: Không có đơn thuốc nào cho bệnh nhân.
- 500 Internal Server Error: Lỗi từ máy chủ.

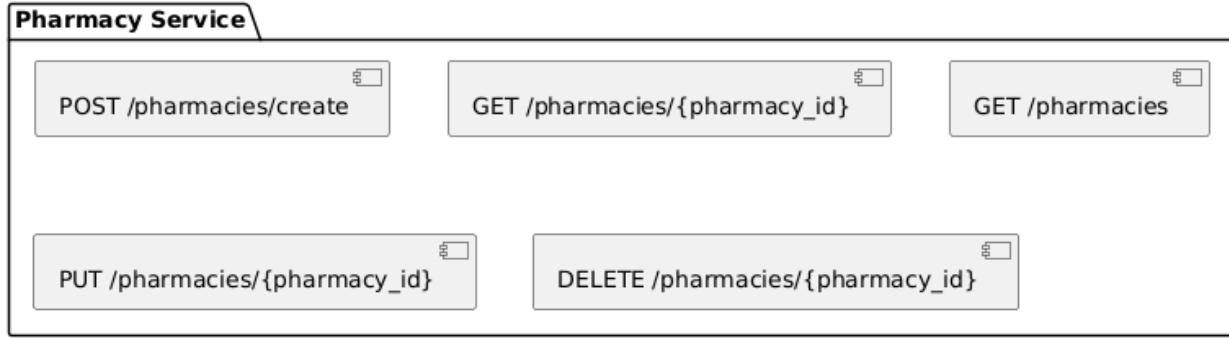
4. PUT /prescriptions/{prescription_id}

- **Mô tả:** Cập nhật thông tin đơn thuốc theo prescription_id.
- **Tham số yêu cầu:**
 - prescription_id (String): ID của đơn thuốc cần cập nhật.
 - medications (Array): Danh sách các thuốc và liều lượng mới.
 - prescription_date (String): Ngày cấp đơn thuốc mới.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin đơn thuốc đã cập nhật.
 - 404 Not Found: Không tìm thấy đơn thuốc với prescription_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /prescriptions/{prescription_id}

- **Mô tả:** Xóa đơn thuốc theo prescription_id.
- **Tham số yêu cầu:**
 - prescription_id (String): ID của đơn thuốc cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, đơn thuốc đã bị xóa.
 - 404 Not Found: Không tìm thấy đơn thuốc với prescription_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.5 Pharmacy Service API Design



1. POST /pharmacies/create

- **Mô tả:** Tạo một cửa hàng thuốc mới.
- **Tham số yêu cầu:**
 - name (String): Tên cửa hàng thuốc.
 - address (String): Địa chỉ của cửa hàng thuốc.
 - contact_info (String): Thông tin liên lạc của cửa hàng thuốc.
 - opening_hours (String): Giờ mở cửa của cửa hàng.
- **Mã trạng thái:**
 - 201 Created: Cửa hàng thuốc được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /pharmacies/{pharmacy_id}

- **Mô tả:** Lấy thông tin chi tiết của một cửa hàng thuốc theo pharmacy_id.
- **Tham số yêu cầu:**
 - pharmacy_id (String): ID của cửa hàng thuốc cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của cửa hàng thuốc.
 - 404 Not Found: Không tìm thấy cửa hàng thuốc với pharmacy_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /pharmacies

- **Mô tả:** Lấy danh sách tất cả các cửa hàng thuốc.
- **Tham số yêu cầu:** Không có tham số yêu cầu.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các cửa hàng thuốc.
 - 500 Internal Server Error: Lỗi từ máy chủ.

4. PUT /pharmacies/{pharmacy_id}

- **Mô tả:** Cập nhật thông tin của một cửa hàng thuốc theo pharmacy_id.
- **Tham số yêu cầu:**
 - pharmacy_id (String): ID của cửa hàng thuốc cần cập nhật.
 - name (String): Tên cửa hàng thuốc mới.
 - address (String): Địa chỉ mới của cửa hàng.
 - contact_info (String): Thông tin liên lạc mới của cửa hàng.
 - opening_hours (String): Giờ mở cửa mới của cửa hàng.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của cửa hàng thuốc đã cập nhật.
 - 404 Not Found: Không tìm thấy cửa hàng thuốc với pharmacy_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /pharmacies/{pharmacy_id}

- **Mô tả:** Xóa cửa hàng thuốc theo pharmacy_id.
- **Tham số yêu cầu:**
 - pharmacy_id (String): ID của cửa hàng thuốc cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, cửa hàng thuốc đã bị xóa.
 - 404 Not Found: Không tìm thấy cửa hàng thuốc với pharmacy_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.6 Billing Service API Design



1. POST /billing/create

- **Mô tả:** Tạo một hóa đơn mới cho bệnh nhân hoặc khách hàng.
- **Tham số yêu cầu:**
 - patient_id (String): ID bệnh nhân hoặc khách hàng.
 - amount (Decimal): Số tiền cần thanh toán.
 - billing_date (String, ISO 8601): Ngày lập hóa đơn.
 - status (String): Trạng thái thanh toán của hóa đơn (ví dụ: "Pending", "Paid").
- **Mã trạng thái:**
 - 201 Created: Hóa đơn được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /billing/{billing_id}

- **Mô tả:** Lấy thông tin chi tiết của một hóa đơn theo billing_id.
- **Tham số yêu cầu:**
 - billing_id (String): ID của hóa đơn cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của hóa đơn.
 - 404 Not Found: Không tìm thấy hóa đơn với billing_id đã cho.

- 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /billing

- **Mô tả:** Lấy danh sách tất cả các hóa đơn.
- **Tham số yêu cầu:** Không có tham số yêu cầu.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các hóa đơn.
 - 500 Internal Server Error: Lỗi từ máy chủ.

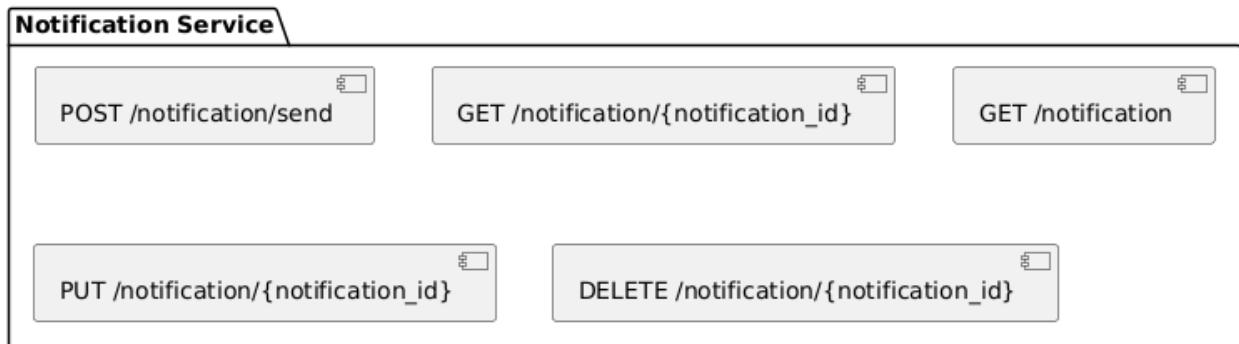
4. PUT /billing/{billing_id}

- **Mô tả:** Cập nhật thông tin của một hóa đơn theo billing_id.
- **Tham số yêu cầu:**
 - billing_id (String): ID của hóa đơn cần cập nhật.
 - amount (Decimal): Số tiền cần thanh toán (có thể thay đổi).
 - status (String): Trạng thái thanh toán mới của hóa đơn (ví dụ: "Paid", "Pending").
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin hóa đơn đã cập nhật.
 - 404 Not Found: Không tìm thấy hóa đơn với billing_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /billing/{billing_id}

- **Mô tả:** Xóa một hóa đơn theo billing_id.
- **Tham số yêu cầu:**
 - billing_id (String): ID của hóa đơn cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, hóa đơn đã bị xóa.
 - 404 Not Found: Không tìm thấy hóa đơn với billing_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.7 Notification Service API Design



1. POST /notification/send

- **Mô tả:** Gửi thông báo cho người dùng hoặc hệ thống.
- **Tham số yêu cầu:**
 - recipient_id (String): ID người nhận thông báo (có thể là người dùng hoặc nhóm người dùng).
 - message (String): Nội dung thông báo.
 - type (String): Loại thông báo (ví dụ: "Email", "SMS", "Push Notification").
 - status (String): Trạng thái gửi thông báo (ví dụ: "Pending", "Sent", "Failed").
- **Mã trạng thái:**
 - 201 Created: Thông báo được gửi thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /notification/{notification_id}

- **Mô tả:** Lấy thông tin chi tiết của một thông báo theo notification_id.
- **Tham số yêu cầu:**
 - notification_id (String): ID của thông báo cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của thông báo.
 - 404 Not Found: Không tìm thấy thông báo với notification_id đã cho.

- 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /notification

- **Mô tả:** Lấy danh sách tất cả các thông báo.
- **Tham số yêu cầu:** Không có tham số yêu cầu.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các thông báo.
 - 500 Internal Server Error: Lỗi từ máy chủ.

4. PUT /notification/{notification_id}

- **Mô tả:** Cập nhật trạng thái của thông báo theo notification_id.
- **Tham số yêu cầu:**
 - notification_id (String): ID của thông báo cần cập nhật.
 - status (String): Trạng thái mới của thông báo (ví dụ: "Pending", "Sent", "Failed").
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của thông báo đã cập nhật.
 - 404 Not Found: Không tìm thấy thông báo với notification_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /notification/{notification_id}

- **Mô tả:** Xóa một thông báo theo notification_id.
- **Tham số yêu cầu:**
 - notification_id (String): ID của thông báo cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, thông báo đã bị xóa.
 - 404 Not Found: Không tìm thấy thông báo với notification_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.9 Insurance Service API Design



1. POST /insurance/create

- **Mô tả:** Tạo một hồ sơ bảo hiểm mới cho người dùng.
- **Tham số yêu cầu:**
 - user_id (String): ID người dùng mà hồ sơ bảo hiểm được tạo cho họ.
 - insurance_type (String): Loại bảo hiểm (ví dụ: "Health", "Life", "Vehicle").
 - start_date (Date): Ngày bắt đầu bảo hiểm.
 - end_date (Date): Ngày kết thúc bảo hiểm.
 - coverage_amount (Number): Số tiền bảo hiểm.
 - status (String): Trạng thái của bảo hiểm (ví dụ: "Active", "Inactive").
- **Mã trạng thái:**
 - 201 Created: Hồ sơ bảo hiểm được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /insurance/{insurance_id}

- **Mô tả:** Lấy thông tin chi tiết của một hồ sơ bảo hiểm theo insurance_id.
- **Tham số yêu cầu:**
 - insurance_id (String): ID của hồ sơ bảo hiểm cần lấy thông tin.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của hồ sơ bảo hiểm.

- 404 Not Found: Không tìm thấy hồ sơ bảo hiểm với insurance_id đã cho.
- 500 Internal Server Error: Lỗi từ máy chủ.

3. GET /insurance/user/{user_id}

- **Mô tả:** Lấy danh sách tất cả các hồ sơ bảo hiểm của một người dùng.
- **Tham số yêu cầu:**
 - user_id (String): ID người dùng cần lấy danh sách bảo hiểm.
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các hồ sơ bảo hiểm của người dùng.
 - 404 Not Found: Không tìm thấy người dùng với user_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

4. PUT /insurance/{insurance_id}

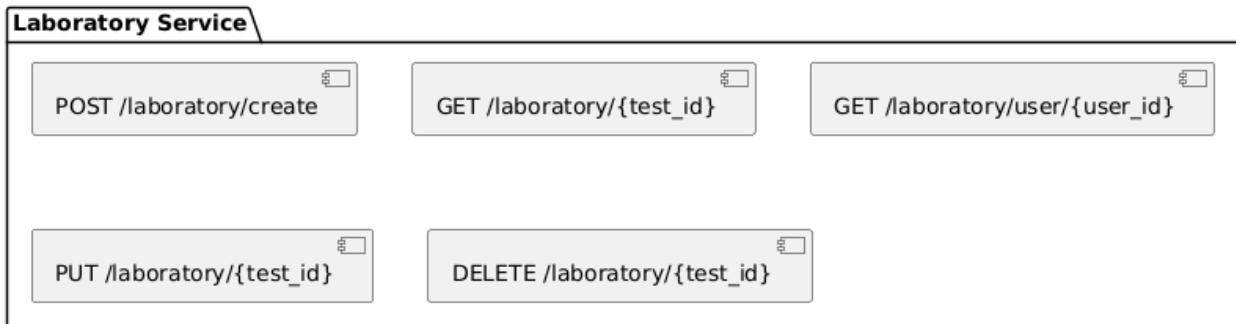
- **Mô tả:** Cập nhật thông tin của một hồ sơ bảo hiểm theo insurance_id.
- **Tham số yêu cầu:**
 - insurance_id (String): ID của hồ sơ bảo hiểm cần cập nhật.
 - status (String): Trạng thái mới của bảo hiểm (ví dụ: "Active", "Inactive").
 - coverage_amount (Number): Cập nhật số tiền bảo hiểm.
 - end_date (Date): Cập nhật ngày kết thúc bảo hiểm (nếu có).
- **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của hồ sơ bảo hiểm đã cập nhật.
 - 404 Not Found: Không tìm thấy hồ sơ bảo hiểm với insurance_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /insurance/{insurance_id}

- **Mô tả:** Xóa một hồ sơ bảo hiểm theo insurance_id.
- **Tham số yêu cầu:**
 - insurance_id (String): ID của hồ sơ bảo hiểm cần xóa.

- **Mã trạng thái:**
 - 200 OK: Thành công, hồ sơ bảo hiểm đã bị xóa.
 - 404 Not Found: Không tìm thấy hồ sơ bảo hiểm với insurance_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.3.10 Laboratory Service API Design



1. POST /laboratory/create

- **Mô tả:** Tạo một yêu cầu xét nghiệm mới trong hệ thống.
- **Tham số yêu cầu:**
 - user_id (String): ID người dùng cần xét nghiệm.
 - test_type (String): Loại xét nghiệm (ví dụ: "Blood Test", "Urine Test", "X-ray").
 - test_date (Date): Ngày yêu cầu xét nghiệm.
 - status (String): Trạng thái của yêu cầu xét nghiệm (ví dụ: "Pending", "Completed").
 - priority (String): Mức độ ưu tiên của xét nghiệm (ví dụ: "High", "Medium", "Low").
- **Mã trạng thái:**
 - 201 Created: Yêu cầu xét nghiệm đã được tạo thành công.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2. GET /laboratory/{test_id}

- **Mô tả:** Lấy thông tin chi tiết của một yêu cầu xét nghiệm theo test_id.

- **Tham số yêu cầu:**
 - test_id (String): ID của yêu cầu xét nghiệm cần lấy thông tin.
 - **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của yêu cầu xét nghiệm.
 - 404 Not Found: Không tìm thấy yêu cầu xét nghiệm với test_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.
- 3. GET /laboratory/user/{user_id}**
- **Mô tả:** Lấy danh sách tất cả các yêu cầu xét nghiệm của người dùng.
 - **Tham số yêu cầu:**
 - user_id (String): ID người dùng cần lấy danh sách yêu cầu xét nghiệm.
 - **Mã trạng thái:**
 - 200 OK: Thành công, trả về danh sách các yêu cầu xét nghiệm của người dùng.
 - 404 Not Found: Không tìm thấy người dùng với user_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.
- 4. PUT /laboratory/{test_id}**
- **Mô tả:** Cập nhật trạng thái hoặc thông tin của một yêu cầu xét nghiệm theo test_id.
 - **Tham số yêu cầu:**
 - test_id (String): ID của yêu cầu xét nghiệm cần cập nhật.
 - status (String): Trạng thái mới của yêu cầu xét nghiệm (ví dụ: "Completed", "In Progress").
 - result (String): Kết quả của xét nghiệm.
 - **Mã trạng thái:**
 - 200 OK: Thành công, trả về thông tin của yêu cầu xét nghiệm đã cập nhật.
 - 404 Not Found: Không tìm thấy yêu cầu xét nghiệm với test_id đã cho.
 - 400 Bad Request: Dữ liệu không hợp lệ.
 - 500 Internal Server Error: Lỗi từ máy chủ.

5. DELETE /laboratory/{test_id}

- **Mô tả:** Xóa một yêu cầu xét nghiệm theo test_id.
- **Tham số yêu cầu:**
 - test_id (String): ID của yêu cầu xét nghiệm cần xóa.
- **Mã trạng thái:**
 - 200 OK: Thành công, yêu cầu xét nghiệm đã bị xóa.
 - 404 Not Found: Không tìm thấy yêu cầu xét nghiệm với test_id đã cho.
 - 500 Internal Server Error: Lỗi từ máy chủ.

2.4 Database design

2.4.1 Database Design – User Service (PostgreSQL)

1. Kiến trúc tổng quan

User Service bao gồm hai bảng chính:

- **users:** Lưu thông tin xác thực và quyền của người dùng.
- **user_profiles:** Lưu thông tin chi tiết hồ sơ cá nhân liên kết với người dùng.

Hai bảng này tuân thủ nguyên tắc **separation of concerns:** bảng users dùng để xử lý xác thực, phân quyền (authentication & authorization), trong khi user_profiles phục vụ các dữ liệu hồ sơ có thể mở rộng linh hoạt.

2. Mô tả bảng chi tiết

2.1 users table

Trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PK, not null, default gen_random_uuid()	Định danh duy nhất cho người dùng
username	VARCHAR(255)	unique, not null	Tên đăng nhập hoặc email/số điện thoại
email	VARCHAR(255)	unique, not null	Email người dùng

Trường	Kiểu dữ liệu	Ràng buộc	Mô tả
password	TEXT	not null	Mật khẩu đã được mã hóa
role	ENUM	not null	Vai trò (PATIENT, DOCTOR, ...)
is_active	BOOLEAN	default true	Tình trạng hoạt động của tài khoản
created_at	TIMESTAMP	default now()	Ngày tạo
updated_at	TIMESTAMP	default now()	Ngày cập nhật gần nhất

Ghi chú: role nên được định nghĩa dưới dạng PostgreSQL ENUM để đảm bảo tính toàn vẹn dữ liệu và tránh sai sót.

2.2 user_profiles table

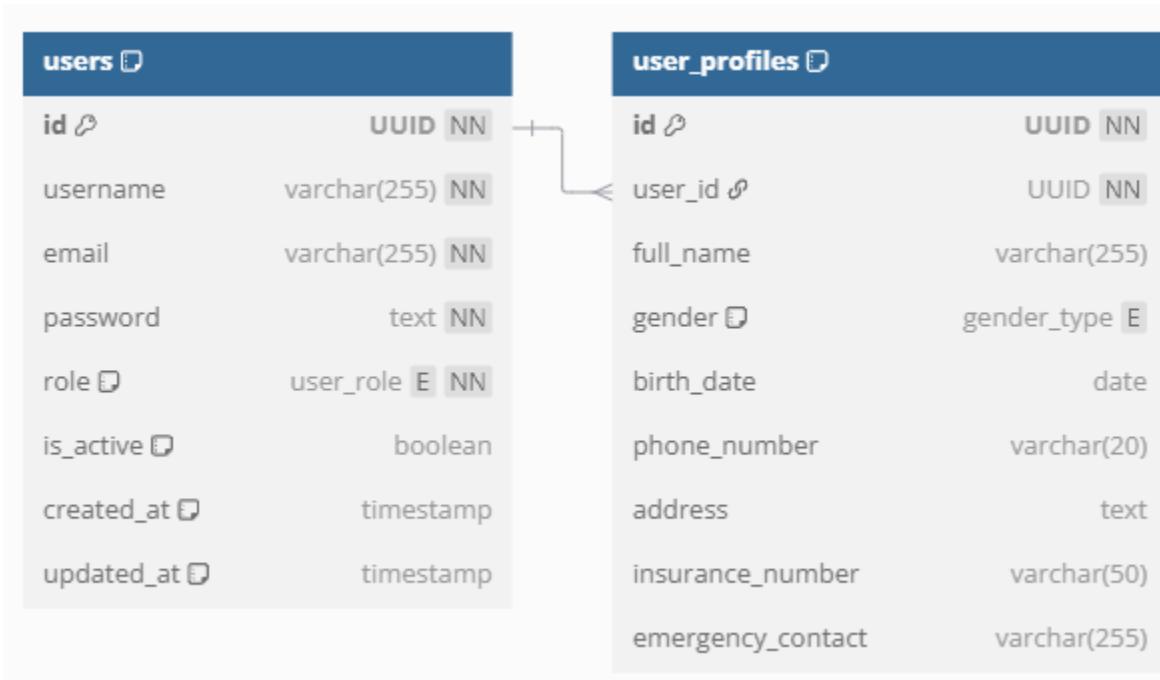
Trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PK, not null, default gen_random_uuid()	Định danh hồ sơ
user_id	UUID	FK -> users(id), unique, not null	Ràng buộc 1-1 với bảng users
full_name	VARCHAR(255)		Họ và tên người dùng
gender	ENUM		Giới tính (MALE, FEMALE, OTHER)
birth_date	DATE		Ngày sinh
phone_number	VARCHAR(20)		Số điện thoại
address	TEXT		Địa chỉ người dùng
insurance_number	VARCHAR(50)		Mã số bảo hiểm y tế
emergency_contact	VARCHAR(255)		Thông tin người liên hệ khẩn cấp

3. Quan hệ giữa các bảng

- **One-to-One:** users \leftrightarrow user_profiles

Mỗi người dùng có duy nhất một hồ sơ cá nhân. Điều này được đảm bảo bằng ràng buộc **UNIQUE** trên user_id trong user_profiles.

4. DBML



2.4.2 Database Design – Appointment Service (MySQL)

1. Kiến trúc tổng quan

Appointment Service quản lý thông tin lịch hẹn giữa bệnh nhân và bác sĩ. Hệ thống được thiết kế gồm hai bảng chính:

- **appointments:** Lưu trữ lịch hẹn và trạng thái hiện tại.
- **appointment_histories:** Ghi nhận lại lịch sử thay đổi của từng lịch hẹn (reschedule, cancel, confirm...).

Hệ thống hỗ trợ truy vết lịch sử thay đổi, đảm bảo tính minh bạch và phù hợp với các yêu cầu nghiệp vụ trong môi trường y tế.

2. Mô tả bảng chi tiết

2.1 appointments table

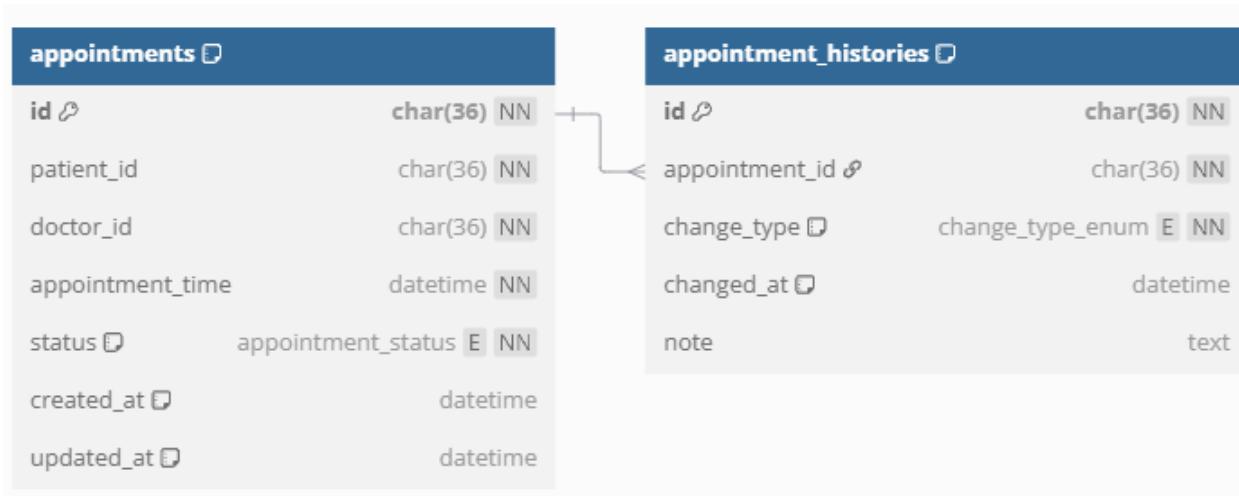
Trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	CHAR(36)	PK, NOT NULL	UUID định danh duy nhất
patient_id	CHAR(36)	NOT NULL, FK → users(id)	Khóa ngoại đến bệnh nhân
doctor_id	CHAR(36)	NOT NULL, FK → users(id)	Khóa ngoại đến bác sĩ
appointment_time	DATETIME	NOT NULL	Thời gian đặt lịch hẹn
status	ENUM	NOT NULL	Trạng thái: PENDING, CONFIRMED, COMPLETED, CANCELLED
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Ngày tạo lịch hẹn
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Ngày cập nhật lịch hẹn

Lưu ý: Vì MySQL không hỗ trợ native UUID, ta sử dụng CHAR(36).

2.2 appointment_histories table

Trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	CHAR(36)	PK, NOT NULL	UUID định danh
appointment_id	CHAR(36)	NOT NULL, FK → appointments(id)	Khóa ngoại đến lịch hẹn
change_type	ENUM	NOT NULL	Loại thay đổi: SCHEDULED, RESCHEDULED, CANCELLED
changed_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Thời điểm thay đổi
note	TEXT	NULL	Ghi chú (nếu có)

3. DBML (Database Markup Language)



2.4.3 Database Design – Medical Record Service (MongoDB)

1. Kiến trúc tổng quan

Medical Record Service quản lý toàn bộ hồ sơ bệnh án, bao gồm thông tin chẩn đoán, đơn thuốc, kết quả xét nghiệm và ghi chú của bác sĩ. Hệ thống sử dụng MongoDB để tận dụng tính linh hoạt trong lưu trữ dữ liệu bán cấu trúc (ví dụ: kết quả xét nghiệm dạng văn bản, ảnh, PDF...).

MongoDB phù hợp trong trường hợp này vì:

- Có thể lưu trữ danh sách kết quả xét nghiệm dưới dạng mảng hoặc tài liệu nhúng.
- Hỗ trợ mở rộng theo chiều ngang tốt cho dữ liệu phi cấu trúc.
- Phù hợp với đặc tính thay đổi linh hoạt của dữ liệu bệnh án.

2. Mô tả tài liệu chi tiết

2.1 Collection: medical_records

```
{  
  _id: ObjectId,  
  patient_id: UUID,  
  doctor_id: UUID,
```

```
appointment_id: UUID,
```

```
diagnosis: String,
```

```
prescription: [String],
```

```
test_results: [
```

```
{
```

```
    test_type: String,
```

```
    test_date: Date,
```

```
    result: String,
```

```
    image: String
```

```
}
```

```
],
```

```
notes: String,
```

```
created_at: Date,
```

```
updated_at: Date
```

```
}
```

- **prescription:** Danh sách thuốc và chỉ định điều trị (mỗi phần tử là một chuỗi).
- **test_results:** Dữ liệu xét nghiệm có thể được nhúng trực tiếp (embedded documents) vì thường gắn liền với hồ sơ.
- **image:** Đường dẫn ảnh nếu có (có thể liên kết với hệ thống lưu trữ như MinIO hoặc Cloud Storage).

2.2 (Tùy chọn) Collection: test_results (nếu muốn tách riêng)

Nếu xét nghiệm lớn, có thể tách riêng bảng test_results:

```
{
```

```
    _id: ObjectId,
```

```
    medical_record_id: ObjectId,
```

```
    test_type: String,
```

```

    test_date: Date,
    result: String,
    image: String
}

```

Tuy nhiên, với trường hợp bình thường, nên nhúng kết quả vào medical_records để giảm chi phí join và tối ưu truy xuất.

3. Mô hình DBML tương đương (dùng để mô tả cấu trúc logic, không triển khai được trên MongoDB)

Chỉ sử dụng để hình dung mối quan hệ giữa các collection, không dùng để migrate schema.

medical_records		test_results	
_id	objectid	_id	objectid
patient_id	uuid	medical_record_id	objectid
doctor_id	uuid	test_type	string
appointment_id	uuid	test_date	datetime
diagnosis	string	result	string
prescription	string[]	image	string
test_results	json[]		
notes	string		
created_at	datetime		
updated_at	datetime		

2.4.4 Database Design – Prescription Service (PostgreSQL)

1. Kiến trúc tổng quan

Prescription Service chịu trách nhiệm lưu trữ thông tin đơn thuốc được kê bởi bác sĩ sau mỗi lần khám. Dữ liệu liên kết chặt chẽ với bệnh nhân, bác sĩ và cuộc hẹn, đồng thời bao gồm danh sách các loại thuốc và hướng dẫn sử dụng chi tiết.

2. Thiết kế bảng dữ liệu

Prescription Service bao gồm hai bảng chính:

- prescriptions: chứa thông tin tổng quan của đơn thuốc.
- medicines: chứa danh sách thuốc cụ thể trong từng đơn.

2.1 Bảng prescriptions

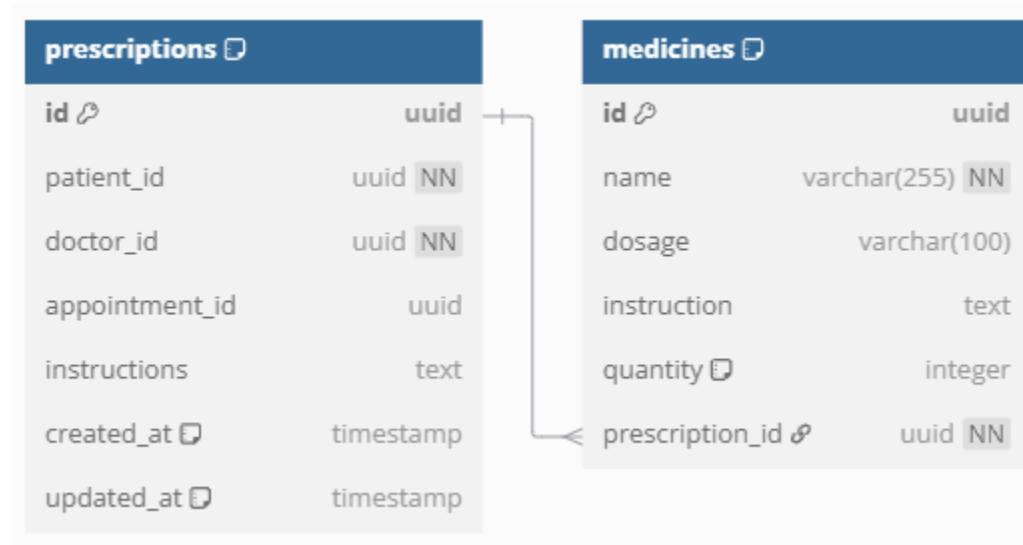
Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PRIMARY KEY	Định danh đơn thuốc
patient_id	UUID	NOT NULL, FK -> users(id)	Khóa ngoại đến người bệnh
doctor_id	UUID	NOT NULL, FK -> users(id)	Khóa ngoại đến bác sĩ
appointment_id	UUID	FK -> appointments(id)	Khóa ngoại đến lịch hẹn
instructions	TEXT		Chỉ định của bác sĩ
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Thời điểm tạo đơn thuốc
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Thời điểm cập nhật đơn thuốc

Ghi chú: Có thể dùng trigger hoặc application logic để cập nhật updated_at.

2.2 Bảng medicines

Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PRIMARY KEY	Định danh thuốc
name	VARCHAR(255)	NOT NULL	Tên thuốc
dosage	VARCHAR(100)		Liều dùng thuốc
instruction	TEXT		Cách sử dụng
quantity	INTEGER	CHECK (quantity >= 0)	Số lượng thuốc
prescription_id	UUID	NOT NULL, FK -> prescriptions(id)	Khóa ngoại đến đơn thuốc

3. Mô hình DBML



2.4.5 Database Design – Pharmacy Service (MySQL)

1. Kiến trúc tổng quan

Pharmacy Service chịu trách nhiệm quản lý dữ liệu liên quan đến thuốc, bao gồm tồn kho, thông tin chi tiết, giá cả và các đơn thuốc tương ứng. Nó đóng vai trò là nguồn dữ liệu trung tâm để phục vụ các yêu cầu từ hệ thống đơn thuốc, kho thuốc và xử lý đơn hàng thuốc.

2. Thiết kế bảng dữ liệu

Pharmacy Service bao gồm hai bảng chính:

- medicines: quản lý thông tin thuốc và tồn kho.
 - prescription_medicines: bảng liên kết giữa đơn thuốc và thuốc (nhiều-nhiều).
-

2.1 Bảng medicines

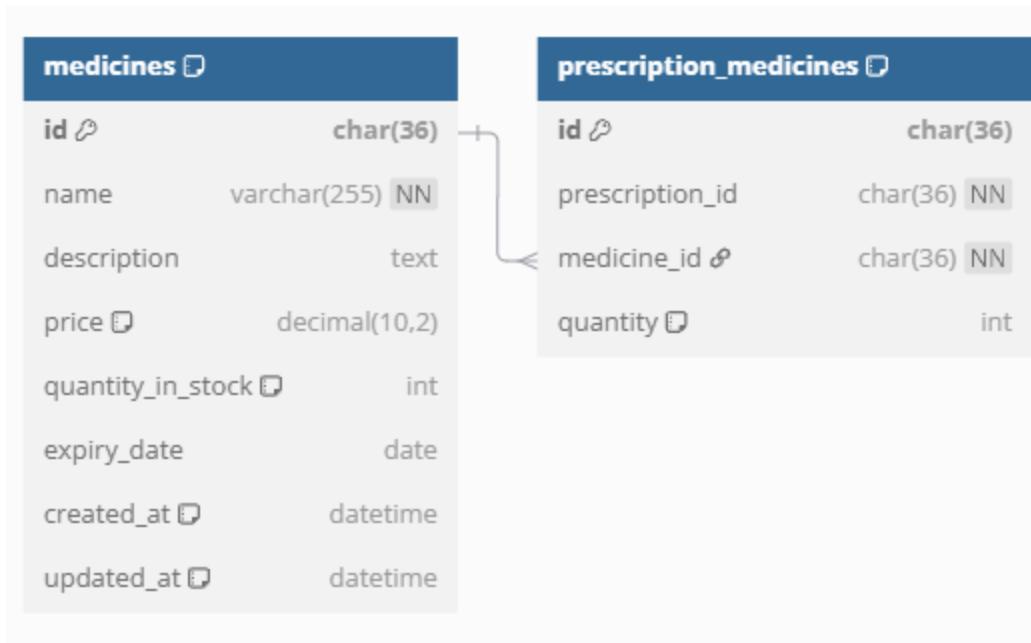
Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	CHAR(36)	PRIMARY KEY	Định danh duy nhất cho thuốc (UUID)

Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
name	VARCHAR(255)	NOT NULL	Tên thuốc
description	TEXT		Mô tả chi tiết về thuốc
price	DECIMAL(10, 2)	CHECK (price >= 0)	Giá thuốc
quantity_in_stock	INT	CHECK (quantity_in_stock >= 0)	Số lượng thuốc hiện có trong kho
expiry_date	DATE		Ngày hết hạn của thuốc
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Ngày thêm vào hệ thống
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Ngày cập nhật gần nhất

2.2 Bảng prescription_medicines

Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	CHAR(36)	PRIMARY KEY	Định danh quan hệ đơn thuốc - thuốc (UUID)
prescription_id	CHAR(36)	NOT NULL, FK → prescriptions(id)	Khóa ngoại đến bảng Prescription
medicine_id	CHAR(36)	NOT NULL, FK → medicines(id)	Khóa ngoại đến bảng Medicine
quantity	INT	CHECK (quantity > 0)	Số lượng thuốc tương ứng trong đơn thuốc

3. Mô hình DBML



2.4.6 Database Design – Billing Service (PostgreSQL)

1. Kiến trúc tổng quan

Billing Service chịu trách nhiệm quản lý hóa đơn và thanh toán cho các dịch vụ y tế. Dữ liệu bao gồm thông tin hóa đơn, trạng thái thanh toán, hạn thanh toán và lịch sử thanh toán của bệnh nhân.

2. Thiết kế bảng dữ liệu

Billing Service bao gồm hai bảng chính:

- bills: quản lý thông tin hóa đơn.
 - payments: quản lý thông tin thanh toán cho mỗi hóa đơn.
-

2.1 Bảng bills

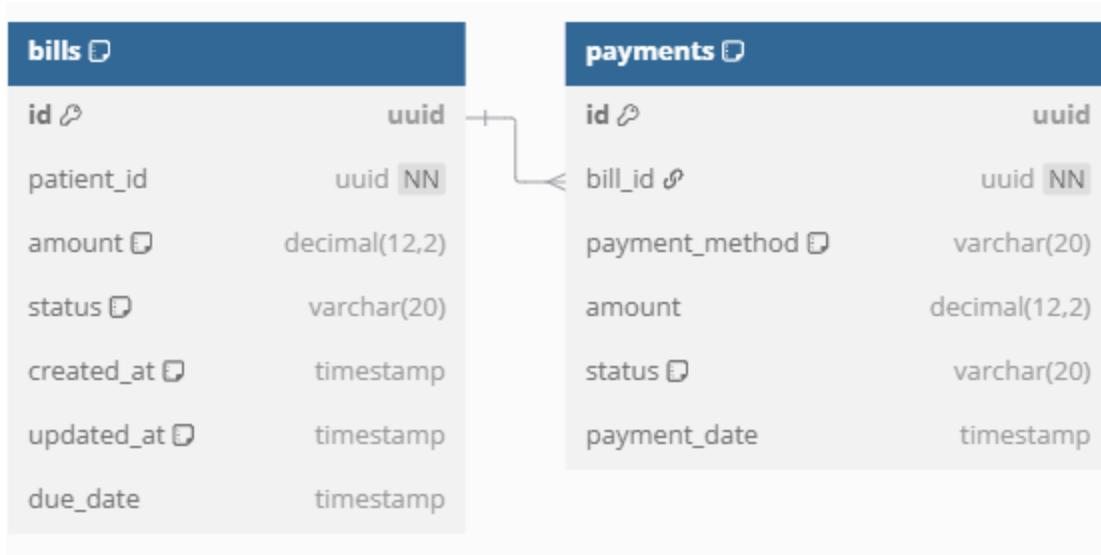
Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PRIMARY KEY	Định danh duy nhất cho hóa đơn

Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
patient_id	UUID	NOT NULL, FK → users(id)	Khóa ngoại đến bệnh nhân (User)
amount	DECIMAL(12, 2)	CHECK (amount >= 0)	Tổng số tiền hóa đơn
status	VARCHAR(20)	CHECK (status IN ('Pending', 'Paid', 'Cancelled'))	Trạng thái hóa đơn
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Thời gian tạo hóa đơn
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Thời gian cập nhật hóa đơn
due_date	TIMESTAMP		Ngày hết hạn thanh toán

2.2 Bảng payments

Tên trường	Kiểu dữ liệu	Ràng buộc	Mô tả
id	UUID	PRIMARY KEY	Định danh thanh toán
bill_id	UUID	NOT NULL, FK → bills(id)	Khóa ngoại đến hóa đơn
payment_method	VARCHAR(20)	CHECK (payment_method IN ('CreditCard', 'Cash', 'Insurance'))	Phương thức thanh toán
amount	DECIMAL(12, 2)	CHECK (amount >= 0)	Số tiền đã thanh toán
status	VARCHAR(20)	CHECK (status IN ('Completed', 'Pending', 'Failed'))	Trạng thái thanh toán
payment_date	TIMESTAMP		Ngày thực hiện thanh toán

3. Mô hình DBML



2.4.7 Database Design – Notification Service (MongoDB)

1. Kiến trúc tổng quan

Notification Service chịu trách nhiệm gửi và theo dõi các thông báo đến người dùng hệ thống (bao gồm bệnh nhân, bác sĩ, dược sĩ, v.v.). Dữ liệu được lưu trữ dưới dạng tài liệu MongoDB, cho phép linh hoạt trong việc mở rộng loại thông báo và nội dung.

2. Thiết kế collection dữ liệu

Notification Service chỉ bao gồm một collection chính là **notifications**.

2.1 Collection notifications

Trường	Kiểu dữ liệu	Mô tả
id	ObjectId	Định danh duy nhất cho thông báo
user_id	UUID	Khóa ngoại đến người dùng nhận thông báo
message	String	Nội dung thông báo gửi tới người dùng
type	String (Enum)	Loại thông báo: Appointment, Payment, LabResult, v.v.
status	String (Enum)	Trạng thái gửi: Sent, Pending, Failed

Trường	Kiểu dữ liệu	Mô tả
read	Boolean	Trạng thái đã đọc: true hoặc false
created_at	DateTime	Thời điểm tạo thông báo
updated_at	DateTime	Thời điểm cập nhật thông báo

3. Mô hình JSON mẫu

```
{
  "_id": ObjectId("661f7f3b4e2a3f3b12345678"),
  "user_id": "f3c91e7a-0b98-4563-8725-a23f432aa887",
  "message": "Lịch hẹn của bạn với bác sĩ Nguyễn Văn A đã được xác nhận.",
  "type": "Appointment",
  "status": "Sent",
  "read": false,
  "created_at": "2025-04-14T10:00:00Z",
  "updated_at": "2025-04-14T10:00:00Z"
}
```

2.4.8 Database Design – Insurance Service (PostgreSQL)

1. Kiến trúc tổng quan

Insurance Service chịu trách nhiệm quản lý thông tin yêu cầu bồi thường bảo hiểm và chính sách bảo hiểm của bệnh nhân. Dữ liệu được lưu trữ trong PostgreSQL với các quan hệ rõ ràng giữa bệnh nhân và các đơn yêu cầu cũng như hợp đồng bảo hiểm.

2. Thiết kế bảng dữ liệu

Insurance Service bao gồm hai bảng chính: insurance_claims và insurance_policies.

2.1 Bảng insurance_claims

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho yêu cầu bảo hiểm
patient_id	UUID	Khóa ngoại đến bệnh nhân (bảng users)
claim_number	String	Mã số yêu cầu bảo hiểm (unique)
amount_claimed	Decimal	Số tiền yêu cầu bảo hiểm
status	Enum	Trạng thái: Pending, Approved, Rejected
date_of_claim	DateTime	Ngày yêu cầu bảo hiểm
created_at	DateTime	Thời điểm tạo yêu cầu
updated_at	DateTime	Thời điểm cập nhật gần nhất

Gợi ý ENUM cho cột status:

sql

CopyEdit

```
CREATE TYPE claim_status AS ENUM ('Pending', 'Approved', 'Rejected');
```

2.2 Bảng insurance_policies

Trường	Kiểu dữ liệu	Mô tả
id	UUID	Định danh duy nhất cho chính sách bảo hiểm
policy_number	String	Số hợp đồng bảo hiểm (unique)
patient_id	UUID	Khóa ngoại đến bệnh nhân (bảng users)
coverage_amount	Decimal	Số tiền bảo hiểm được chi trả tối đa
policy_type	Enum	Loại bảo hiểm: Health, Life, etc.
start_date	DateTime	Ngày bắt đầu hiệu lực hợp đồng
end_date	DateTime	Ngày hết hạn hợp đồng bảo hiểm

Trường	Kiểu dữ liệu	Mô tả
status	Enum	Trạng thái: Active, Expired, Suspended

3. Quan hệ giữa các bảng

- Một User có thể có nhiều insurance_policies.
- Một User cũng có thể gửi nhiều insurance_claims.
- Có thể thêm ràng buộc giữa insurance_claims.claim_number và các công ty bảo hiểm nếu hệ thống mở rộng quản lý bên thứ ba.

4. DBML

insurance_claims		insurance_policies	
id	UUID	id	UUID
patient_id	UUID NN	policy_number	varchar NN
claim_number	varchar NN	patient_id	UUID NN
amount_claimed	decimal NN	coverage_amount	decimal NN
status	claim_status E NN	policy_type	policy_type_enum E NN
date_of_claim	timestamp NN	start_date	timestamp NN
created_at	timestamp	end_date	timestamp NN
updated_at	timestamp	status	policy_status_enum E NN

2.3.10 Laboratory Service Database Design

Laboratory Service là dịch vụ quản lý các đơn yêu cầu xét nghiệm và kết quả xét nghiệm của bệnh nhân. Dịch vụ này bao gồm hai bảng chính:

1. **TestOrder:** Lưu trữ thông tin về các đơn yêu cầu xét nghiệm, bao gồm thông tin bệnh nhân, bác sĩ yêu cầu, loại xét nghiệm và trạng thái của đơn yêu cầu xét nghiệm.
 2. **TestResult:** Lưu trữ kết quả xét nghiệm, bao gồm dữ liệu kết quả và trạng thái của kết quả.
-

Tables and Descriptions

2.1 TestOrder

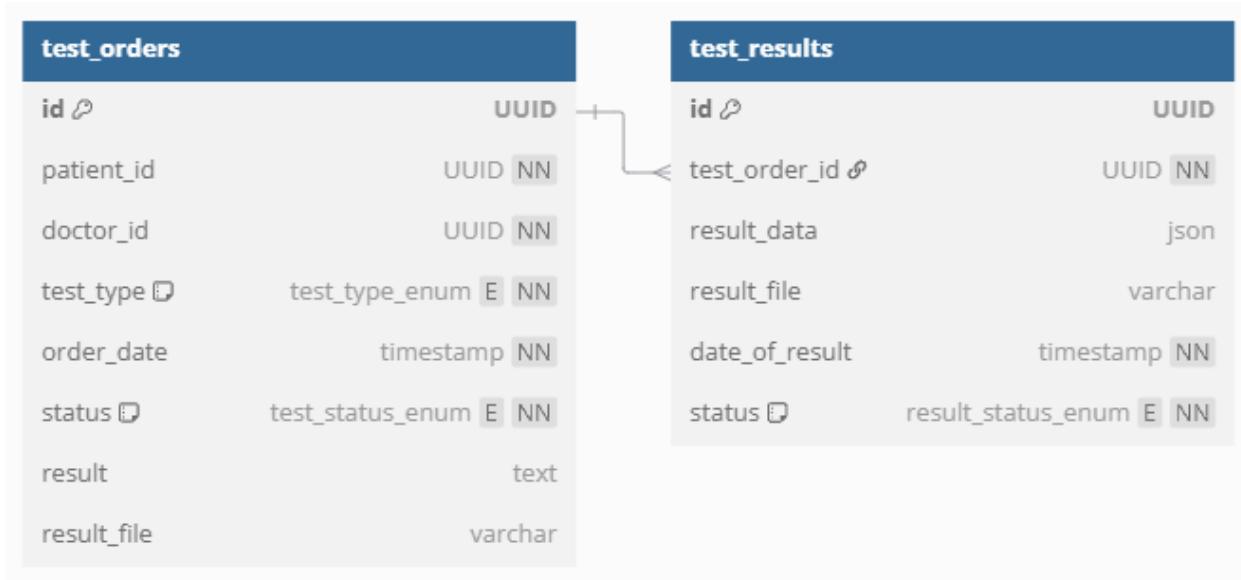
TestOrder bảng này lưu trữ thông tin chi tiết về mỗi đơn yêu cầu xét nghiệm, bao gồm các thông tin như người yêu cầu xét nghiệm, loại xét nghiệm, trạng thái, và kết quả (nếu có). Các trường chính trong bảng này bao gồm:

- **id**: Định danh duy nhất cho đơn yêu cầu xét nghiệm.
- **patient_id**: Khóa ngoại liên kết đến bảng người bệnh (User).
- **doctor_id**: Khóa ngoại liên kết đến bác sĩ yêu cầu xét nghiệm.
- **test_type**: Loại xét nghiệm yêu cầu (ví dụ: máu, nước tiểu, X-quang, siêu âm, v.v.).
- **order_date**: Ngày tạo đơn yêu cầu xét nghiệm.
- **status**: Trạng thái hiện tại của đơn yêu cầu xét nghiệm (ví dụ: PENDING, IN PROGRESS, COMPLETED).
- **result**: Kết quả xét nghiệm (nếu có) sau khi xét nghiệm hoàn thành.
- **result_file**: Đường dẫn tới file kết quả xét nghiệm, có thể là PDF hoặc hình ảnh.

2.2 TestResult

TestResult bảng này lưu trữ thông tin chi tiết về kết quả của mỗi xét nghiệm. Các trường chính bao gồm:

- **id**: Định danh duy nhất cho kết quả xét nghiệm.
- **test_order_id**: Khóa ngoại liên kết đến bảng **TestOrder**, đại diện cho đơn yêu cầu xét nghiệm mà kết quả này liên quan.
- **result_data**: Dữ liệu kết quả xét nghiệm, có thể là văn bản, số liệu hoặc hình ảnh, tùy thuộc vào loại xét nghiệm.
- **result_file**: Đường dẫn tới file kết quả xét nghiệm, ví dụ như file PDF hoặc hình ảnh.
- **date_of_result**: Ngày kết quả xét nghiệm có sẵn.
- **status**: Trạng thái của kết quả xét nghiệm (PENDING, COMPLETED, REJECTED).



2.5 Kết luận: CHƯƠNG 2 - THIẾT KẾ HỆ THỐNG THƯƠNG MẠI ĐIỆN TỬ VỚI MICRO-SERVICES VÀ DJANGO

Trong Chương 2, chúng ta đã khám phá chi tiết về thiết kế kiến trúc và các thành phần chính của một hệ thống thương mại điện tử được xây dựng bằng microservices và Django. Chương này nhằm cung cấp một nền tảng vững chắc để hiểu cách chia nhỏ một hệ thống phức tạp thành các dịch vụ nhỏ hơn, có thể quản lý, mở rộng, và cung cấp tính linh hoạt cho sự phát triển trong tương lai.

Các điểm nổi bật chính:

- Kiến trúc Microservices:** Hệ thống thương mại điện tử được thiết kế theo kiến trúc microservices để tăng cường tính mô-đun, khả năng mở rộng và bảo trì. Mỗi dịch vụ (như User Service, Appointment Service, Medical Record Service,...) đều có thể triển khai độc lập và giao tiếp giữa các dịch vụ này diễn ra thông qua RESTful APIs. Thiết kế này cho phép mỗi dịch vụ có thể phát triển độc lập mà không ảnh hưởng đến toàn bộ hệ thống, đồng thời cung cấp sự phục hồi bằng cách cài đặt sự cố trong các dịch vụ riêng biệt.
- API Gateway:** API Gateway đóng vai trò là điểm trung gian xử lý các yêu cầu từ người dùng và điều phối các yêu cầu này đến các dịch vụ tương ứng. Với sự trợ giúp của API Gateway, hệ thống có thể dễ dàng quản lý lưu lượng truy cập và bảo mật thông qua các cơ chế như xác thực và phân quyền. Ngoài ra, API Gateway còn hỗ trợ việc giám sát và log các hoạt động hệ thống, giúp tăng cường khả năng kiểm tra và duy trì hệ thống.
- Chiến lược Cơ sở Dữ liệu:** Mỗi dịch vụ trong hệ thống sử dụng một cơ sở dữ liệu riêng biệt, không chia sẻ dữ liệu với các dịch vụ khác. Điều này đảm bảo tính độc lập giữa các

dịch vụ, giúp hệ thống dễ dàng mở rộng và bảo trì. Cơ sở dữ liệu được lựa chọn phù hợp với yêu cầu của từng dịch vụ, ví dụ: PostgreSQL cho các dịch vụ yêu cầu tính nhất quán cao, MySQL cho các dịch vụ cần khả năng truy vấn nhanh và MongoDB cho các dịch vụ yêu cầu lưu trữ dữ liệu phi cấu trúc.

4. **Bảo mật với JWT:** Để đảm bảo an toàn thông tin người dùng và các giao dịch trong hệ thống, chúng tôi đã sử dụng JSON Web Tokens (JWT) để xác thực người dùng và cấp quyền truy cập cho từng dịch vụ. Việc sử dụng JWT giúp tăng cường bảo mật cho hệ thống, đồng thời duy trì tính linh hoạt trong việc phân quyền và kiểm soát truy cập.
5. **Triển khai và Docker:** Hệ thống được triển khai dưới dạng các container Docker, với mỗi dịch vụ và cơ sở dữ liệu đều chạy trong một container riêng biệt. Docker Compose được sử dụng để quản lý việc khởi động và dừng các dịch vụ, giúp đơn giản hóa quá trình triển khai và kiểm tra trong môi trường phát triển và thử nghiệm.
6. **Thiết kế API:** Các API được thiết kế để phục vụ các chức năng chính của hệ thống, từ việc quản lý người dùng, đặt lịch khám, quản lý hồ sơ bệnh án, cho đến việc xử lý các yêu cầu bảo hiểm và thanh toán. Các API này đảm bảo tính nhất quán trong việc cung cấp dịch vụ và dễ dàng mở rộng để phục vụ nhu cầu trong tương lai.

Tóm lại:

Chương 2 đã xây dựng một hệ thống thương mại điện tử phức tạp với kiến trúc microservices, nơi mỗi dịch vụ được thiết kế để độc lập và tương tác thông qua các API. Các quyết định về công nghệ, bảo mật, cơ sở dữ liệu, và triển khai đều nhằm tối ưu hóa khả năng mở rộng và bảo trì hệ thống. Các thành phần này phối hợp chặt chẽ để tạo ra một hệ thống vững mạnh, dễ dàng thích nghi với sự thay đổi và phát triển trong tương lai.

Chapter 3: AI IN HEALTH CARE SYSTEMS

3.1 Ứng dụng của Trí tuệ nhân tạo trong Y tế

Trí tuệ nhân tạo (AI) đang trở thành một lực lượng cách mạng trong ngành y tế, mang lại những giải pháp sáng tạo để nâng cao hiệu quả chẩn đoán, điều trị, chăm sóc bệnh nhân và quản lý hành chính. Các ứng dụng của AI trong y tế có thể được chia thành các lĩnh vực chính sau:

3.1.1 Chẩn đoán và hình ảnh y học

Các thuật toán AI, đặc biệt là các mô hình học sâu (deep learning), đã đạt được thành công lớn trong việc phân tích hình ảnh y học như X-quang, CT, MRI và siêu âm. Hệ thống AI có thể phát hiện các bất thường như khối u, gãy xương, nhiễm trùng với độ chính xác cao, thậm chí trong một số trường hợp còn vượt qua các bác sĩ chuyên khoa. Ví dụ, AI hiện đang được sử dụng trong việc sàng lọc ung thư vú, ung thư phổi, và bệnh võng mạc đái tháo đường.

3.1.2 Hỗ trợ quyết định lâm sàng

AI giúp các bác sĩ đưa ra quyết định chính xác hơn bằng cách phân tích dữ liệu lớn về bệnh án, kết quả xét nghiệm, và nghiên cứu y khoa. Các hệ thống hỗ trợ ra quyết định lâm sàng (CDSS) có thể đề xuất phác đồ điều trị, phát hiện sớm các dấu hiệu bất thường, hoặc cảnh báo về các tương tác thuốc nguy hiểm.

3.1.3 Phát triển thuốc và cá thể hóa điều trị

AI đang rút ngắn đáng kể thời gian nghiên cứu và phát triển thuốc bằng cách dự đoán cấu trúc phân tử, mô phỏng phản ứng sinh học và xác định mục tiêu điều trị tiềm năng. Ngoài ra, AI còn giúp cá thể hóa điều trị bằng cách đề xuất các phác đồ phù hợp với bộ gen, thể trạng và tiền sử bệnh của từng bệnh nhân.

3.1.4 Robot phẫu thuật

Robot phẫu thuật tích hợp AI giúp các ca phẫu thuật trở nên chính xác hơn, ít xâm lấn hơn và an toàn hơn. Robot có thể hỗ trợ trong các thao tác phức tạp, giảm thiểu sai sót và rút ngắn thời gian hồi phục cho bệnh nhân.

3.1.5 Chăm sóc sức khỏe từ xa và theo dõi bệnh nhân

AI được ứng dụng trong các thiết bị đeo tay, cảm biến y tế và hệ thống chăm sóc từ xa để giám sát liên tục các chỉ số sức khỏe như nhịp tim, huyết áp, đường huyết. Dữ liệu được phân tích theo thời gian thực để phát hiện sớm nguy cơ và can thiệp kịp thời.

3.1.6 Quản lý bệnh viện và hành chính

AI cũng giúp các cơ sở y tế tối ưu hóa hoạt động quản lý, từ việc tự động hóa quy trình đăng ký, lưu trữ hồ sơ bệnh án điện tử, đến quản lý chuỗi cung ứng và dự đoán nhu cầu sử dụng giường bệnh.

3.2 Kỹ thuật Học sâu (Deep Learning)

Học sâu (Deep Learning) là một nhánh quan trọng của trí tuệ nhân tạo, được xây dựng dựa trên các mạng nơ-ron nhân tạo nhiều lớp. Trong y tế, các kỹ thuật học sâu đã chứng minh được khả năng vượt trội trong việc xử lý dữ liệu phức tạp như hình ảnh y học, chuỗi thời gian sinh học, và dữ liệu gen.

3.2.1 Mạng Nơ-ron Tích chập (Convolutional Neural Networks - CNN)

CNN là nền tảng chính trong các ứng dụng phân tích hình ảnh y học. Nhờ khả năng tự động trích xuất đặc trưng từ hình ảnh, CNN được dùng để phát hiện các bệnh lý trên X-quang, CT, MRI, và hình ảnh siêu âm. Ví dụ, CNN giúp chẩn đoán sớm ung thư phổi qua hình ảnh CT với độ chính xác cao.

3.2.2 Mạng Nơ-ron Tái hiện (Recurrent Neural Networks - RNN)

RNN và các biến thể như LSTM (Long Short-Term Memory) rất hiệu quả trong việc xử lý chuỗi thời gian, chẳng hạn như theo dõi nhịp tim, điện tâm đồ (ECG) và dữ liệu huyết áp theo thời gian. RNN cho phép dự đoán sớm các biến chứng hoặc cơn đau tim dựa trên chuỗi dữ liệu liên tục.

3.2.3 Mạng Sinh đối Khóa (Generative Adversarial Networks - GAN)

GAN được ứng dụng trong việc tạo ra hình ảnh y học tổng hợp để tăng cường dữ liệu huấn luyện cho các mô hình AI, đặc biệt hữu ích khi dữ liệu thực tế bị hạn chế. Ngoài ra, GAN còn hỗ trợ tái tạo hình ảnh với độ phân giải cao, giúp bác sĩ quan sát chi tiết hơn.

3.2.4 Mạng Biến áp (Transformers)

Transformers, ban đầu nổi tiếng trong xử lý ngôn ngữ tự nhiên (NLP), hiện nay đang được ứng dụng mạnh mẽ trong y học, từ phân tích hồ sơ bệnh án điện tử đến dự đoán chẩn đoán đa bệnh lý. Các mô hình như BERT hay GPT được điều chỉnh để xử lý dữ liệu lâm sàng hiệu quả hơn.

3.2.5 Học sâu đa phương thức (Multimodal Deep Learning)

Trong y tế, dữ liệu rất đa dạng: hình ảnh, văn bản, chỉ số sinh học, và dữ liệu gen. Kỹ thuật học sâu đa phương thức giúp kết hợp nhiều nguồn dữ liệu để nâng cao độ chính xác trong chẩn đoán và cá thể hóa điều trị.

3.2.6 Học chuyển giao (Transfer Learning)

Với học chuyển giao, các mô hình học sâu được huấn luyện trên tập dữ liệu lớn (như ImageNet) có thể được tinh chỉnh (fine-tune) để áp dụng cho các bài toán y tế, giúp tiết kiệm chi phí và thời gian huấn luyện khi dữ liệu y tế thường bị giới hạn.

3.3 Ứng dụng Chatbot trong Y tế

Trong kỷ nguyên chuyển đổi số, chatbot đã trở thành một công cụ quan trọng trong lĩnh vực chăm sóc sức khỏe. Chatbot là các chương trình máy tính có khả năng giao tiếp với con người thông qua ngôn ngữ tự nhiên, giúp tự động hóa các tác vụ tư vấn, hỗ trợ và quản lý bệnh nhân. Đặc biệt, trong bối cảnh đại dịch COVID-19, chatbot đã chứng minh vai trò to lớn trong việc sàng lọc triệu chứng, cung cấp thông tin chính xác, và giảm tải cho các cơ sở y tế.

3.3.1 Các Ứng dụng Chính của Chatbot trong Y tế

- **Tư vấn và chẩn đoán sơ bộ:** Chatbot có thể thu thập thông tin về triệu chứng từ người bệnh, phân tích và gợi ý những bệnh lý có thể gặp, đồng thời khuyến cáo người bệnh nên gặp bác sĩ trong các trường hợp nghiêm trọng.
- **Nhắc nhở dùng thuốc:** Nhiều chatbot hỗ trợ bệnh nhân trong việc tuân thủ liệu trình điều trị bằng cách gửi thông báo nhắc nhở uống thuốc đúng giờ, đúng liều.
- **Hỗ trợ đặt lịch hẹn:** Chatbot giúp bệnh nhân dễ dàng đặt lịch khám, kiểm tra tình trạng lịch hẹn, hoặc hủy lịch mà không cần gọi điện thoại trực tiếp đến bệnh viện.
- **Chăm sóc bệnh nhân mạn tính:** Với những bệnh lý như tiểu đường, tăng huyết áp, hoặc tim mạch, chatbot đóng vai trò theo dõi các chỉ số sức khỏe và cung cấp lời khuyên chăm sóc hàng ngày.
- **Hỗ trợ sức khỏe tinh thần:** Các chatbot như Woebot hay Wysa được thiết kế để hỗ trợ người dùng vượt qua stress, lo âu, và trầm cảm thông qua các bài tập tâm lý và trò chuyện an ủi.

3.3.2 Công nghệ và Mô hình Chatbot Y tế

Các chatbot trong y tế sử dụng công nghệ **Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP)** để hiểu và phản hồi người dùng một cách tự nhiên nhất. Một số chatbot hiện đại còn ứng dụng **học sâu (Deep Learning)** để nâng cao độ chính xác trong hiểu ý người bệnh.

Các nền tảng phổ biến hỗ trợ xây dựng chatbot y tế bao gồm:

- **Dialogflow** của Google
- **Rasa** (mã nguồn mở)
- **Microsoft Bot Framework**

Kiến trúc tổng quát của một chatbot y tế bao gồm:

- Lớp giao tiếp với người dùng (qua website, ứng dụng, mạng xã hội)

- Bộ xử lý ngôn ngữ tự nhiên (NLP)
- Hệ thống cơ sở dữ liệu lưu trữ hồ sơ bệnh nhân và lịch sử trò chuyện
- Mô hình AI hỗ trợ phân tích và ra quyết định.

3.3.3 Lợi ích của Chatbot trong Y tế

- **Giảm tải cho nhân viên y tế:** Chatbot xử lý hàng trăm yêu cầu cùng lúc, giúp bác sĩ tập trung vào các ca bệnh nặng.
- **Phục vụ liên tục 24/7:** Người bệnh có thể nhận hỗ trợ vào bất kỳ thời điểm nào, kể cả ngoài giờ hành chính.
- **Tăng cường khả năng tiếp cận y tế:** Đặc biệt tại các khu vực nông thôn, vùng sâu, nơi thiếu nhân lực y tế.
- **Cá nhân hóa chăm sóc sức khỏe:** Dựa trên dữ liệu lịch sử, chatbot có thể đưa ra lời khuyên phù hợp với từng cá nhân.

3.3.4 Thách thức và Giới hạn

Mặc dù tiềm năng lớn, chatbot y tế cũng đối mặt với nhiều thách thức:

- **Độ chính xác:** Chẩn đoán sơ bộ của chatbot không thể thay thế hoàn toàn ý kiến chuyên môn từ bác sĩ.
- **Bảo mật dữ liệu:** Thông tin y tế cá nhân rất nhạy cảm, đòi hỏi các hệ thống chatbot phải đảm bảo tiêu chuẩn bảo mật cao.
- **Hạn chế trong xử lý trường hợp phức tạp:** Chatbot phù hợp với các yêu cầu đơn giản, nhưng khi gặp ca bệnh nguy hiểm, việc chuyển giao cho nhân viên y tế là cần thiết.

3.3.5 Xu hướng và Tương lai

Trong tương lai, chatbot y tế sẽ ngày càng mạnh mẽ hơn nhờ:

- **Tích hợp với thiết bị đeo thông minh (wearable devices):** Giúp theo dõi nhịp tim, huyết áp, lượng đường huyết theo thời gian thực.

- **Ứng dụng AI mạnh mẽ:** Nhờ các mô hình như GPT-4, chatbot sẽ hiểu ngôn ngữ tự nhiên sâu sắc hơn.
- **Phát triển các giải pháp chăm sóc tại nhà (homecare):** Hỗ trợ bệnh nhân điều trị và theo dõi bệnh từ xa, giảm thiểu chi phí và thời gian di chuyển.

3.3.6 Triển khai

a) Train model

```
{
  "intents": [
    {
      "question": "Hi there",
      "answer": "Hi there, how can I help?",
      "url": "",
      "tags": ["greeting"]
    },
    {
      "question": "Bye",
      "answer": "Have a nice day",
      "url": "",
      "tags": ["goodbye"]
    },
    {
      "question": "Thanks",
      "answer": "You're Welcome",
      "url": "",
      "tags": ["thanks"]
    },
    {
      "answer": "stopping smoking is about will power and being steadfast. you can stop safely by having bupropion or nicotine patch cover initially in",
      "question": "how do i stop smoking now",
      "url": "http://ehealthforum.com/health/stop-smoking-question-t462882.html",
      "tags": [
        "addiction",
        "stop smoking"
      ]
    },
    {
      "answer": "hello this sounds quite unfamiliar that due to no reason at this age you have no cycle for last three months. pregnancy is surely a re",
      "question": "i had a tubaligation 4 years ago and also have a minor case of endometriosis i am only 27 and my period stopped completely 3 months"
    }
  ]
}
```

```
# Step 1: Import necessary libraries
import json
import numpy as np
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM
from keras import layers
# from keras.preprocessing.text import Tokenizer
# from keras.utils import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn.model_selection import train_test_split
# from kerastuner.tuners import RandomSearch
# from kerastuner.engine.hyperparameters import HyperParameters    deltaDC,
from keras_tuner.tuners import RandomSearch
from keras_tuner.engine.hyperparameters import HyperParameters

import pickle
```

Executed at 2025.05.21 15:18:49 in 10ms

```
# Step 2: Download required NLTK resources
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

Executed at 2025.05.21 15:18:50 in 0ms

```
# Step 3: Load stop words and initialize lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

Executed at 2025.05.21 15:18:50 in 4ms

# Step 4: Load and parse the intents file
with open('intents.json') as file:
    data = json.load(file)

Executed at 2025.05.21 15:18:50 in 53ms

# Step 5: Initialize lists to hold training data
training_sentences = []
training_labels = []
labels = []
responses = []

Executed at 2025.05.21 15:18:50 in 4ms

# Step 6: Define preprocessing function
def preprocess_text(text):
    text = re.sub('[^a-zA-Z]', ' ', text) # Remove special characters and numbers
    words = nltk.word_tokenize(text) # Tokenize
    words = [lemmatizer.lemmatize(word.lower()) for word in words if word.lower() not in stop_words] # Lemmatize and remove stopwords
    return ' '.join(words)

Executed at 2025.05.21 15:18:50 in 13ms
```

```
# Step 7: Extract and preprocess data from intents
for intent in data['intents']:
    if len(intent['tags']) > 0:
        training_sentences.append(preprocess_text(intent['question']))
        training_labels.append(intent['tags'][0])
        responses.append(intent['answer'])
        if intent['tags'][0] not in labels:
            labels.append(intent['tags'][0])
```

Executed at 2025.05.21 15:18:53 in 3s 642ms

```
# Step 8: Encode labels
num_classes = len(labels)
lbl_encoder = LabelEncoder()
lbl_encoder.fit(training_labels)
training_labels = lbl_encoder.transform(training_labels)
```

Executed at 2025.05.21 15:18:53 in 21ms

```
# Step 9: Tokenize and pad the input sentences
vocab_size = 2000
embedding_dim = 16
max_len = 200
oov_token = "<OOV>"

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_token)
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(training_sentences)
padded_sequences = pad_sequences(sequences, truncating='post', maxlen=max_len)

# --- Optional Step: Use Keras Tuner for hyperparameter tuning ---
# def build_model(hp):
#     model = Sequential()
#     model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
#     model.add(LSTM(units=hp.Int('units', min_value=50, max_value=150, step=10), return_sequences=True))
#     model.add(LSTM(units=hp.Int('units', min_value=50, max_value=150, step=10)))
#     for i in range(hp.Int('num_layers', 1, 20)):
#         model.add(layers.Dense(units=hp.Int('units_' + str(i), min_value=16, max_value=256, step=16), activation='relu'))
#     model.add(Dense(num_classes, activation='softmax'))
#     model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])), loss='sparse_categorical_crossentropy',
#                   metrics=['accuracy'])
#     return model

Executed at 2025.05.21 15:18:54 in 118ms
```

```
# Step 10: Define and build the LSTM model manually
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(LSTM(110, return_sequences=True))
model.add(LSTM(110))
model.add(Dense(208, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

Executed at 2025.05.21 15:18:54 in 106ms
D:\Workspace\rb\sample-tracker-api\venv\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is
deprecated. Just remove it.
warnings.warn()

# Step 11: Compile the model
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

Executed at 2025.05.21 15:18:54 in 26ms

# Step 12: Show model architecture
model.summary()

Executed at 2025.05.21 15:18:54 in 21ms
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
lstm_1 (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

```
Epoch 22/30
149/149 - 2s - 16ms/step - accuracy: 0.8660 - loss: 0.5375 - val_accuracy: 0.7941 - val_loss: 6.6577
Epoch 23/30
149/149 - 2s - 17ms/step - accuracy: 0.8643 - loss: 0.5359 - val_accuracy: 0.7966 - val_loss: 6.7406
Epoch 24/30
149/149 - 2s - 16ms/step - accuracy: 0.8780 - loss: 0.5004 - val_accuracy: 0.7949 - val_loss: 6.7919
Epoch 25/30
149/149 - 3s - 17ms/step - accuracy: 0.8797 - loss: 0.4766 - val_accuracy: 0.7958 - val_loss: 6.9064
Epoch 26/30
149/149 - 2s - 16ms/step - accuracy: 0.8881 - loss: 0.4544 - val_accuracy: 0.7958 - val_loss: 7.0546
Epoch 27/30
149/149 - 2s - 16ms/step - accuracy: 0.8947 - loss: 0.4217 - val_accuracy: 0.7992 - val_loss: 7.0989
Epoch 28/30
149/149 - 2s - 16ms/step - accuracy: 0.8909 - loss: 0.4300 - val_accuracy: 0.7992 - val_loss: 7.1182
Epoch 29/30
149/149 - 2s - 16ms/step - accuracy: 0.8991 - loss: 0.4062 - val_accuracy: 0.7958 - val_loss: 7.1495
Epoch 30/30
149/149 - 2s - 16ms/step - accuracy: 0.9116 - loss: 0.3589 - val_accuracy: 0.7949 - val_loss: 7.3017
```

```

from matplotlib import pyplot as plt

plt.figure(figsize=(12, 5))

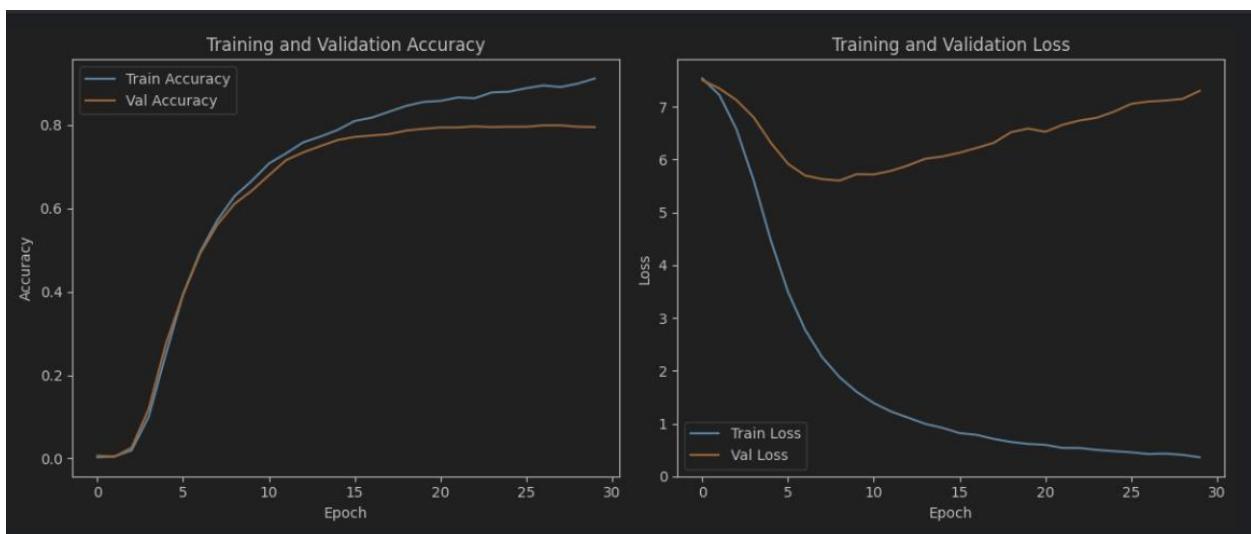
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.tight_layout()
plt.show()

```

Executed at 2025.05.21 18:13:55 in 1s 442ms



```
model.save(f"chat_model_{MODEL_TYPE}.h5")
with open('tokenizer.pickle', 'wb') as f:
    pickle.dump(tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
with open('label_encoder.pickle', 'wb') as f: You, Today + Uncommitted
    pickle.dump(encc, f, protocol=pickle.HIGHEST_PROTOCOL)

print(f"Trained and saved {MODEL_TYPE} model.")
```

Executed at 2025.05.21 18:13:02 in 49ms

b) Triển khai api

```
✓ import json
import numpy as np
import requests
from tensorflow import keras
import pickle
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from flask import Flask, request, jsonify
import nltk

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

app = Flask(__name__)
💡
# Load everything ONCE at startup | deltaDC, Today + add tra
with open("intents.json") as file:
    data = json.load(file)

model = keras.models.load_model('chat_model.h5')

with open('tokenizer.pickle', 'rb') as handle:
    tokenizer = pickle.load(handle)

with open('label_encoder.pickle', 'rb') as enc:
    lbl_encoder = pickle.load(enc)

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
max_len = 200
```

```

# deltaDC

@app.route( rule= '/chat', methods=['POST'])
def chat():
    inp = request.json['input']
    preprocessed_input = preprocess_text(inp)
    sequence = tokenizer.texts_to_sequences([preprocessed_input])
    padded_sequence = keras.preprocessing.sequence.pad_sequences(sequence, truncating='post', maxlen=max_len)
    result = model.predict(padded_sequence)
    probability = float(np.max(result))
    tag = lbl_encoder.inverse_transform([np.argmax(result)])[0]

    if probability < 0.70:
        # Call ChatGPT API if confidence is low
        # print("calling chatgpt")
        # chatgpt_response = call_chatgpt_api(inp)
        # return jsonify({'response': chatgpt_response, 'score': probability})
        return jsonify({'response': "sorry i am not sure", 'score': probability})

    # Otherwise, respond with your intents dataset
    for intent in data['intents']:
        if len(intent['tags']) > 0 and intent['tags'][0] == tag:
            response = {'response': intent['answer'], 'score': probability}
            return jsonify(response)

    # fallback if no match found
    return jsonify({'response': "Sorry, I didn't understand that.", 'score': 0.0})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8090)

```

c) Triển khai trên UI

Welcome, Phạm Đức Chính

Home

HealthCare	Messages	Healthcare Assistant
<ul style="list-style-type: none">DashboardAppointmentsMedical RecordsPrescriptionsInsurancePharmacyMessagesProfileSettings	<p>Search contacts...</p> <p>Healthcare Assistant Just now How can I help you today?</p>	<p>Hello! I'm your Healthcare Assistant. How can I help you today? Just now</p> <p>hi Just now</p> <p>Hi there, how can I help? Just now</p> <p>I hurt my lower back, now I am hunched over and it seems that my ab muscles have changed. Just now</p> <p>it seems you may have a muscle strain in your back muscles. the chiropractor has adjusted your vertebrae which soothed the pain for a while. but the muscles are injured. i recommend doing ice packs and an hot packs on the pin site. rest then deep friction massage. this may need to be assessed by an orthopedist. Just now</p> <p>Pain left breast, cough, 103 fever, cannot sleep on side? Just now</p> <p>hi these may be signs of pneumonitis. get a chest xray done higher antibiotics are indicated if the fever is not reducing. complete blood check up should be done with blood culture to see for any remote infection source. deep breathing exercises will help thanks Just now</p>

CHƯƠNG 4: Code và Triển khai hệ thống Healthcare

1. Công nghệ sử dụng

Hệ thống healthcare được xây dựng với kiến trúc tách biệt giữa frontend và backend, kết hợp nhiều công nghệ hiện đại:

- **Frontend:**
 - Sử dụng **Next.js** – một framework phát triển từ React, hỗ trợ server-side rendering và tối ưu hiệu suất.
 - Giao tiếp với backend thông qua REST API.
- **Backend:**
 - Sử dụng **Django** – framework Python mạnh mẽ, phù hợp với phát triển ứng dụng web nhanh và bảo mật.
 - Mỗi nhóm chức năng chính được xây dựng dưới dạng dịch vụ riêng biệt (service-based design).
- **Triển khai:**
 - **Docker**: dùng để đóng gói các backend service.
 - **Docker Compose**: quản lý và chạy nhiều container đồng thời (multi-service application).
 - Các thành phần chạy:
 - `user_service` và `appointment_service`: được đóng gói và chạy bằng Docker Compose.
 - `frontend` và `chatbot_service`: chạy cục bộ để phục vụ phát triển và kiểm thử.

2. Cấu trúc hệ thống

Hệ thống được chia thành 3 service chính, cùng với frontend:

- **user_service**:
 - Xử lý thông tin người dùng, bao gồm cả bệnh nhân và bác sĩ.
 - Quản lý xác thực, phân quyền người dùng.
- **appointment_service**:
 - Quản lý lịch khám, xử lý logic đặt lịch giữa bác sĩ và bệnh nhân.
- **chatbot_service**:
 - Một AI đơn giản tư vấn sức khỏe cho người dùng thông qua câu hỏi/đáp.
- **frontend**:
 - Giao diện người dùng, kết nối đến các API của backend để hiển thị thông tin và tương tác.

Các dịch vụ backend (user và appointment) hoạt động trong môi trường container riêng biệt nhưng có thể giao tiếp nội bộ thông qua mạng Docker Compose.

3. Các module đã xây dựng

3.1. `user_service` (Django)

- **Module quản lý bác sĩ:**
 - Thêm, sửa, xoá thông tin bác sĩ.
 - Lưu trữ các thông tin chuyên môn: chuyên khoa, lịch khám, thông tin liên hệ.
- **Module quản lý bệnh nhân:**
 - Đăng ký bệnh nhân, cập nhật hồ sơ sức khỏe cá nhân.
 - Quản lý lịch sử khám chữa bệnh.
- **Xác thực và phân quyền:**
 - Hệ thống phân biệt người dùng theo vai trò: bác sĩ, bệnh nhân, quản trị viên.
 - Sử dụng Django authentication và JWT để xác thực.

3.2. `appointment_service` (Django)

- **Quản lý lịch hẹn:**
 - Đặt lịch khám giữa bệnh nhân và bác sĩ, với kiểm tra trùng lặp thời gian.
 - Hủy lịch, cập nhật lịch, xác nhận lịch khám.
 - Lưu trữ trạng thái lịch hẹn: chờ xác nhận, đã xác nhận, đã hoàn thành, đã huỷ.

3.3. `chatbot_service` (Python)

- **Tư vấn sức khỏe cơ bản:**
 - Hệ thống AI đơn giản có khả năng nhận câu hỏi và trả về các gợi ý sức khỏe liên quan.
 - Chạy local, tương tác bằng giao diện dòng lệnh hoặc frontend test.
 - Chưa tích hợp sâu vào hệ thống nhưng có thể mở rộng thành API độc lập sau này.

4. Triển khai hệ thống

4.1. Docker hóa các dịch vụ backend

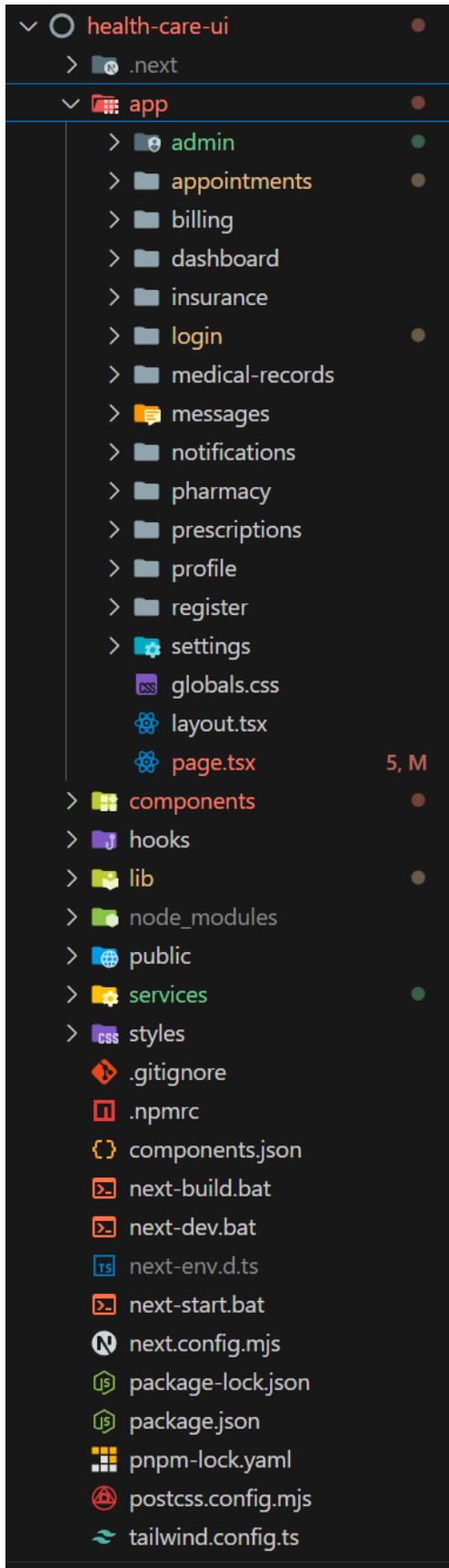
- `user_service` và `appointment_service` được đóng gói thành container riêng biệt.
- Mỗi service có Dockerfile riêng, chứa hướng dẫn cài đặt môi trường, cài đặt thư viện, và chạy server.
- Docker Compose quản lý:
 - Khởi động cả hai service cùng lúc.
 - Kết nối chúng với mạng nội bộ, chia sẻ cơ sở dữ liệu nếu cần.
 - Dùng file `.env` để cấu hình biến môi trường (PORT, DB, SECRET,...).
 -

4.2. Chạy frontend và chatbot local

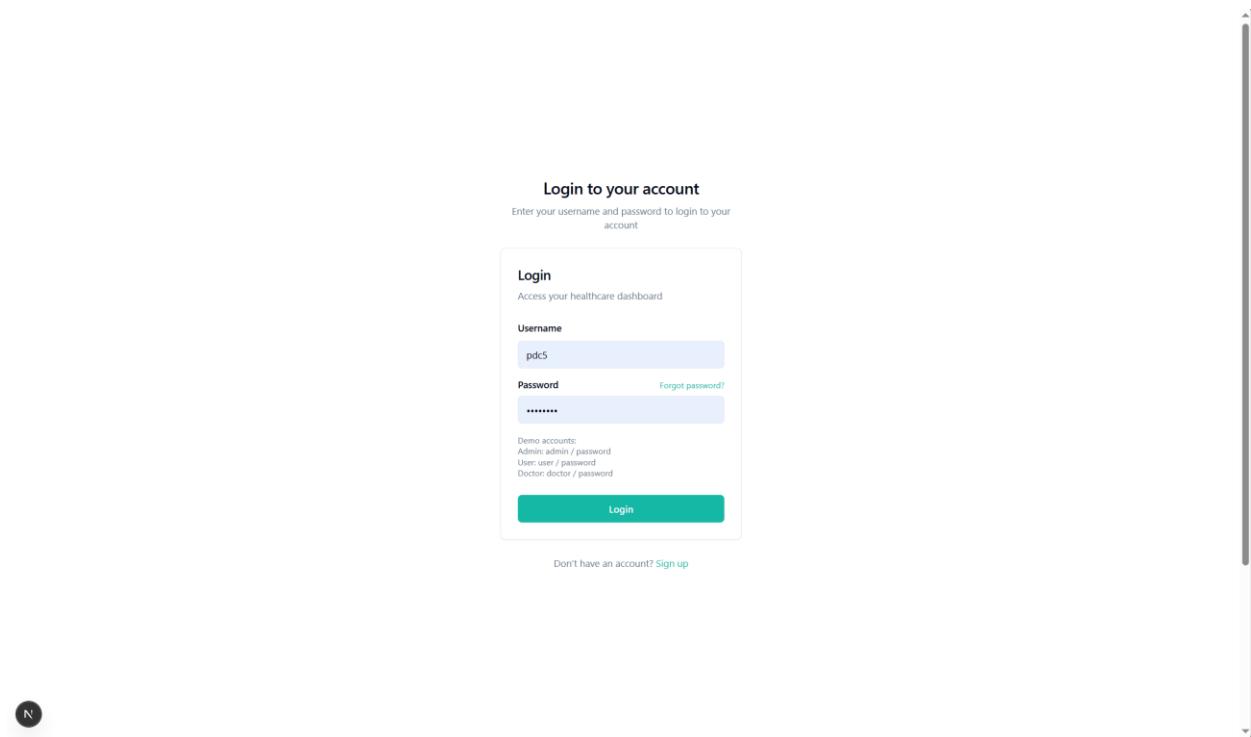
- **Frontend:**
 - Chạy local bằng lệnh `npm run dev` từ thư mục Next.js.
 - Kết nối đến các API đã định nghĩa sẵn trong biến môi trường hoặc qua proxy config.
- **Chatbot:**
 - Chạy local bằng Python (`python chatbot.py`) để kiểm tra tính năng tư vấn.
 - Có thể tương tác qua command line hoặc giao diện frontend kết nối socket/API.

5. Demo

	health_care	Running (4/4)	1.1%	2 seconds ago	
□	user_db 424a59dd8df6	mysql:8.0	Running	0.54% 3306:3306 3 seconds ago	
□	appointment_db 9ba4e2449370	mysql:8.0	Running	0.53% 3307:3307 3 seconds ago	
□	user_service a0b87e448a45	health_care-user_service	Running	0.02% 8000:8000 2 seconds ago	
□	appointment_service 7c43537bd866	health_care-appointment_service	Running	0.01% 8001:8001 2 seconds ago	



Trang login/signup



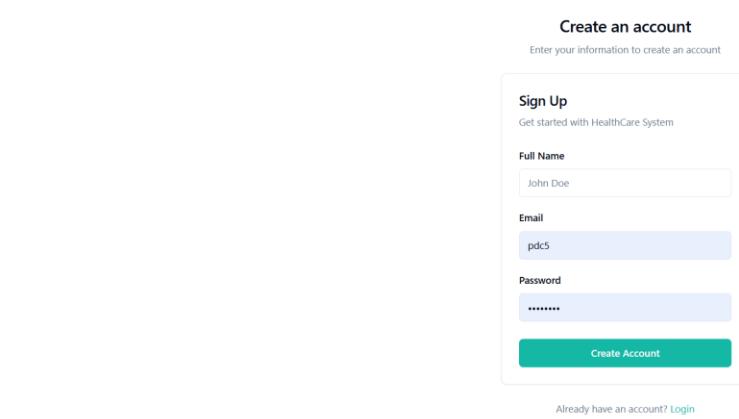
The screenshot shows a web-based login interface. At the top center, it says "Login to your account" with a sub-instruction "Enter your username and password to login to your account". Below this is a "Login" form with the following fields:

- Username:** pdc5
- Password:** (redacted)
- Forgot password?** (link)

Underneath the form, there is a section titled "Demo accounts:" with three entries:

- Admin: admin / password
- User: user / password
- Doctor: doctor / password

A large green "Login" button is at the bottom of the form. Below the form, a small note says "Don't have an account? [Sign up](#)".



The screenshot shows a web-based sign-up interface. At the top center, it says "Create an account" with a sub-instruction "Enter your information to create an account". Below this is a "Sign Up" form with the following fields:

- Full Name:** John Doe
- Email:** pdc5
- Password:** (redacted)

A large green "Create Account" button is at the bottom of the form. Below the form, a small note says "Already have an account? [Login](#)".

Trang dashboard chung sau khi đăng nhập

Welcome, pdc5 (PATIENT)

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Dashboard

Upcoming Appointments
2 Next: May 10, 2023 - Dr. Johnson

Active Prescriptions
4 3 medications due today

Recent Test Results
2 New results from: Blood work, X-ray

Your Doctors
3 Primary care, Cardiologist, Dermatologist

Book Appointment

Upcoming Appointments
Your scheduled appointments for the next 30 days

chinh
laborum anim in
October 12, 2027 03:26 AM
View Details

chinh
laborum anim in
May 30, 2025 09:00 AM
Reason: in-person
Upcoming
View Details

Medication Schedule
Your daily medication reminders

Lisinopril 10mg
8:00 AM - 1 tablet daily

Metformin 500mg
9:00 AM and 7:00 PM - 1 tablet twice daily

Vitamin D 2000 IU
9:00 AM - 1 tablet daily

Login successful
You have been logged in successfully.

Appointment

Welcome, pdc5 (PATIENT)

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Appointments
Manage and schedule your medical appointments

Book New Appointment

Your Appointments
View all your appointments

All Appointments ▾

chinh
laborum anim in
October 12, 2027 03:26 AM
Cancelled
View Details

chinh
laborum anim in
May 30, 2025 09:00 AM
Reason: in-person
Upcoming
View Details

Tạo mới appointment

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Welcome, pdc5 (PATIENT)

HealthCare

- Dashboard
- Appointments**
- Medical Records
- Prescriptions
- Insurance
- Pharmacy
- Messages
- Profile
- Settings

Book Appointment

Schedule a new appointment with a doctor

New Appointment

Fill in the details to schedule your appointment

Select Doctor

chinh - laborum anim in

Next

N

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Welcome, pdc5 (PATIENT)

HealthCare

- Dashboard
- Appointments**
- Medical Records
- Prescriptions
- Insurance
- Pharmacy
- Messages
- Profile
- Settings

Book Appointment

Schedule a new appointment with a doctor

New Appointment

Fill in the details to schedule your appointment

Select Date

May 2025

Su	Mo	Tu	We	Th	Fr	Sa
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Back Next

N

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Welcome, pdc5 (PATIENT)

HealthCare

- Dashboard
- Appointments**
- Medical Records
- Prescriptions
- Insurance
- Pharmacy
- Messages
- Profile
- Settings

Book Appointment
Schedule a new appointment with a doctor

New Appointment
Fill in the details to schedule your appointment

Select Time

<input type="radio"/> 9:00 AM	<input type="radio"/> 9:30 AM	<input type="radio"/> 10:00 AM	<input type="radio"/> 10:30 AM
<input type="radio"/> 11:00 AM	<input type="radio"/> 11:30 AM	<input type="radio"/> 1:00 PM	<input type="radio"/> 1:30 PM
<input type="radio"/> 2:00 PM	<input type="radio"/> 2:30 PM	<input type="radio"/> 3:00 PM	<input type="radio"/> 3:30 PM
<input type="radio"/> 4:00 PM	<input type="radio"/> 4:30 PM		

Back Next

HealthCare Home Dashboard Appointments Medical Records Prescriptions Billing

Welcome, pdc5 (PATIENT)

HealthCare

- Dashboard
- Appointments**
- Medical Records
- Prescriptions
- Insurance
- Pharmacy
- Messages
- Profile
- Settings

Book Appointment
Schedule a new appointment with a doctor

New Appointment
Fill in the details to schedule your appointment

Appointment Type

In-Person Visit

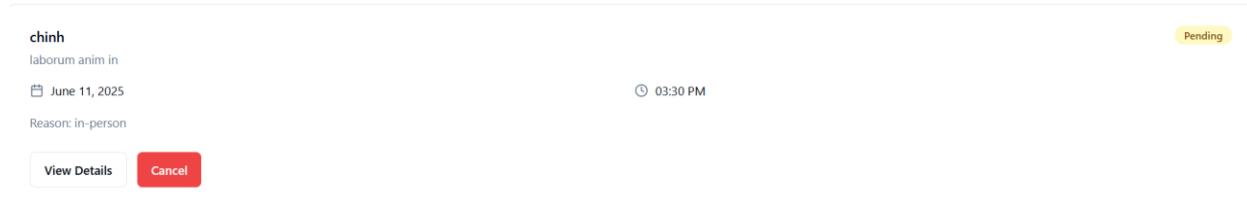
Video Consultation

Phone Consultation

Additional Notes (Optional)

Describe your symptoms or reason for visit

Back Book Appointment



Đăng nhập bằng tài khoản bác sĩ thì thấy appointment tương ứng với bệnh nhân

A screenshot of a web-based patient management system. The left sidebar shows "HealthCare" with "Appointments" selected. The main area is titled "Appointments" and says "Manage and schedule your medical appointments". It has a "Book New Appointment" button. Below is a section titled "Your Appointments" with a "View all your appointments" link and a "Cancelled" button. Three appointment cards are listed:

- pdc5**
Patient ID: acf2c622-8ada-4293-b3b4-f6f59192ac54
Date: October 12, 2027
Time: 03:26 AM
Reason: in-person
Status: Cancelled
Buttons: View Details, Cancel
- pdc5**
Patient ID: acf2c622-8ada-4293-b3b4-f6f59192ac54
Date: May 30, 2025
Time: 09:00 AM
Reason: in-person
Status: Upcoming
Buttons: View Details
- pdc5**
Patient ID: acf2c622-8ada-4293-b3b4-f6f59192ac54
Date: June 11, 2025
Time: 03:30 PM
Reason: in-person
Status: Pending
Buttons: View Details, Cancel

Module PatientManagement, DoctorManagement phải đăng nhập bằng tài khoản admin

HealthCare Admin

- Dashboard
- Patients**
- Doctor
- Appointments
- Medical Records
- Billing
- Insurance
- Pharmacy
- Laboratory
- Notifications
- Reports
- Settings

Patient Management

Manage all patient records in the system

Patients
A list of all patients registered in the system

Patient	Username	Email	Phone	Medical Info	Actions
P No Name Not specified	pdc3	pdc3@gmail.com	Not provided		
P No Name Not specified	pdc5	pdc5@gmail.com	Not provided		
P No Name Not specified	pdc2	pdc2@gmail.com	Not provided		
D ducchinhanh Male	pdc	pdc@gmail.com	0123456789	Blood: A+ Allergies Chronic	

Search patients...

HealthCare Admin

- Dashboard
- Patients**
- Doctor
- Appointments
- Medical Records
- Billing
- Insurance
- Pharmacy
- Laboratory
- Notifications
- Reports
- Settings

Patient Management

Manage all patient records in the system

Patients
A list of all patients registered in the system

Patient	User	Actions
P No Name Not specified	pdc	
P No Name Not specified	pdc	
P No Name Not specified	pdc	
D ducchinhanh Male	pdc	

Patient Details

Comprehensive information about the patient.

ducchinhanh Patient Active User ID: d3872bd8-7787-409a-996d-982a1df21d8c

User Information

Username	pdc
Email	pdc@gmail.com
Full Name	duccinh
Gender	Male
Phone Number	0123456789
Date of Birth	December 13th, 2003
Registration Date	May 28th, 2025

Medical Information

Blood Group	A+
Height	174.00 cm
Weight	70.00 kg

Allergies

Excepiteur

Chronic Diseases

Excepiteur in quis consequat laborum

Current Medications

ad amet eu

Search patients...

Admin Panel - pdc2

HealthCare Admin

Patient Management
Manage all patient records in the system

Patients
A list of all patients registered in the system

P	No Name	Not specified
P	No Name	Not specified
P	No Name	Not specified
D	duchinhh	Male

Edit Patient
Update the patient information in the form below.

User Information

Username*	Email*
pdc	pdc@gmail.com

Medical Information

Blood Group	Height (cm)	Weight (kg)
A+	174.00	70.00

Allergies
Excepteur

Chronic Diseases
Excepteur in quis consequat laborum

Current Medications
ad amet eu

Medical History

Main Site Logout + Add Patient

Search patients...

Actions

Admin Panel - pdc2

HealthCare Admin

Doctor Management
Manage all doctors in the healthcare system

Doctors
A list of all doctors registered in the system

Doctor	Specialization	Email	Phone	License	Actions
C Dr. chinh Male	laborum anim in	pdc4@gmail.com	123123123123	ali 1 years	

Search doctors...

Main Site Logout + Add Doctor

N

Admin Panel - pdc2

Main Site Logout

HealthCare Admin

Doctor Management

Manage all doctors in the healthcare system

Doctors

A list of all doctors registered in the system

Doctor	Specialization	License	Actions
Dr. chinh	Male	Active	<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="View Details"/>

Doctor Details

Comprehensive information about the doctor.

Dr. chinh

laborum anim in

Doctor Active

License: ali

User Information

Username	pdc4
Email	pdc4@gmail.com
Full Name	chinh
Gender	Male
Phone Number	123123123123
Date of Birth	November 11th, 2011
Registration Date	May 28th, 2025

Professional Information

Specialization	laborum anim in
License Number	ali
Qualifications	tempor dolore
Experience	1 years
Consultation Fee	\$1.00
Available Hours	laborum eiusmod

Admin Panel - pdc2

Main Site Logout

HealthCare Admin

Doctor Management

Manage all doctors in the healthcare system

Doctors

A list of all doctors registered in the system

Doctor	Specialization	License	Actions
Dr. chinh	Male	Active	<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="View Details"/>

Edit Doctor

Update the doctor information in the form below.

User Information

Username*	Email*
pdc4	pdc4@gmail.com

Full Name

Gender
Male

Phone Number

Date of Birth

123123123123 11/11/2011

Professional Information

Specialization*	License Number*
laborum anim in	ali

Qualifications

tempor dolore

Years of Experience

Consultation Fee

1 1.00

Available Hours

laborum eiusmod

Chatbot

The screenshot shows a healthcare application interface with a navigation bar at the top. The navigation bar includes links for Home, Dashboard, Appointments, Medical Records, Prescriptions, and Billing. On the far right of the navigation bar are icons for notifications (with a count of 3), settings, and profile.

The main area is titled "Welcome, Phạm Đức Chính". On the left, there is a sidebar with a "Messages" tab selected, showing a list of recent messages. The list includes "Dashboard", "Appointments", "Medical Records", "Prescriptions", "Insurance", "Pharmacy", and "Messages".

The main content area shows a conversation with a "Healthcare Assistant". The healthcare assistant is online and has just sent a message: "Hello! I'm your Healthcare Assistant. How can I help you today? Just now". The patient responds with "hi Just now". The healthcare assistant replies with "Hi there, how can I help? Just now".

The patient then sends a message: "I hurt my lower back, now I am hunched over and it seems that my ab muscles have changed? Just now". The healthcare assistant responds with: "it seems you may have a muscle strain in your back muscles. the chiropractor has adjusted your vertebrae which soothed the pain for a while. but the muscles are injured. i recommend doing ice packs and an hot packs on the pin site. rest then deep friction massage. this may need to be assessed by an orthopedist. Just now".

The patient then sends another message: "Pain left breast, cough, 103 fever, cannot sleep on side? Just now". The healthcare assistant responds with: "hi these may be signs of pneumonitis. get a chest xray done higher antibiotics are indicated if the fever is not reducing. complete blood check up should be done with blood culture to see for any remote infection source. deep breathing exercises will help thanks Just now".

Assignment 3: Chatbot with 4 version

Developing techniques for predicting diabetes (refer to Chap 12, [1])

- kNN
- SVM linear kernel
- SVM RBF kernel
- Logistic Regression

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",
           "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
df = pd.read_csv(url, names=columns)
| You, 2 minutes ago • Uncommitted changes
Executed at 2025.03.13 19:57:35 in 4s 915ms

# Split features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

Executed at 2025.03.13 19:57:35 in 7ms

# Normalize input features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

Executed at 2025.03.13 19:57:35 in 23ms

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

Executed at 2025.03.13 19:57:35 in 11ms

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("\nKNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))

Executed at 2025.03.13 19:57:35 in 33ms

KNN Classification Report:
      precision    recall  f1-score   support

          0       0.74      0.80      0.77      99
          1       0.57      0.49      0.53      55

   accuracy                           0.69      154
  macro avg       0.66      0.64      0.65      154
weighted avg       0.68      0.69      0.68      154

```

```
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train, y_train)
y_pred_svm_linear = svm_linear.predict(X_test)
print("\nSVM (Linear Kernel) Classification Report:")
print(classification_report(y_test, y_pred_svm_linear))
print("SVM (Linear Kernel) Accuracy:", accuracy_score(y_test, y_pred_svm_linear))
```

Executed at 2025.03.13 19:57:35 in 28ms

```
SVM (Linear Kernel) Classification Report:
      precision    recall  f1-score   support

          0       0.81      0.82      0.81      99
          1       0.67      0.65      0.66      55

   accuracy                           0.76      154
  macro avg       0.74      0.74      0.74      154
weighted avg       0.76      0.76      0.76      154
```

SVM (Linear Kernel) Accuracy: 0.7597402597402597

```
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train, y_train)
y_pred_svm_rbf = svm_rbf.predict(X_test)
print("\nSVM (RBF Kernel) Classification Report:")
print(classification_report(y_test, y_pred_svm_rbf))
print("SVM (RBF Kernel) Accuracy:", accuracy_score(y_test, y_pred_svm_rbf))
```

Executed at 2025.03.13 19:57:35 in 27ms

```
SVM (RBF Kernel) Classification Report:
      precision    recall  f1-score   support

          0       0.77      0.82      0.79      99
          1       0.63      0.56      0.60      55

   accuracy                           0.73      154
  macro avg       0.70      0.69      0.70      154
weighted avg       0.72      0.73      0.72      154
```

SVM (RBF Kernel) Accuracy: 0.7272727272727273

```
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)
print("\nLogistic Regression Classification Report:")
print(classification_report(y_test, y_pred_log_reg))
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log_reg))
```

Executed at 2025.03.13 19:57:35 in 20ms

```

Logistic Regression Classification Report:
precision    recall    f1-score   support

          0       0.81      0.80      0.81      99
          1       0.65      0.67      0.66      55

   accuracy                           0.75      154
macro avg       0.73      0.74      0.73      154
weighted avg    0.76      0.75      0.75      154

```

Logistic Regression Accuracy: 0.7532467532467533

```

models = ['KNN', 'SVM (Linear)', 'SVM (RBF)', 'Logistic Regression']
accuracies = [
    accuracy_score(y_test, y_pred_knn),
    accuracy_score(y_test, y_pred_svm_linear),
    accuracy_score(y_test, y_pred_svm_rbf),
    accuracy_score(y_test, y_pred_log_reg)
]

```

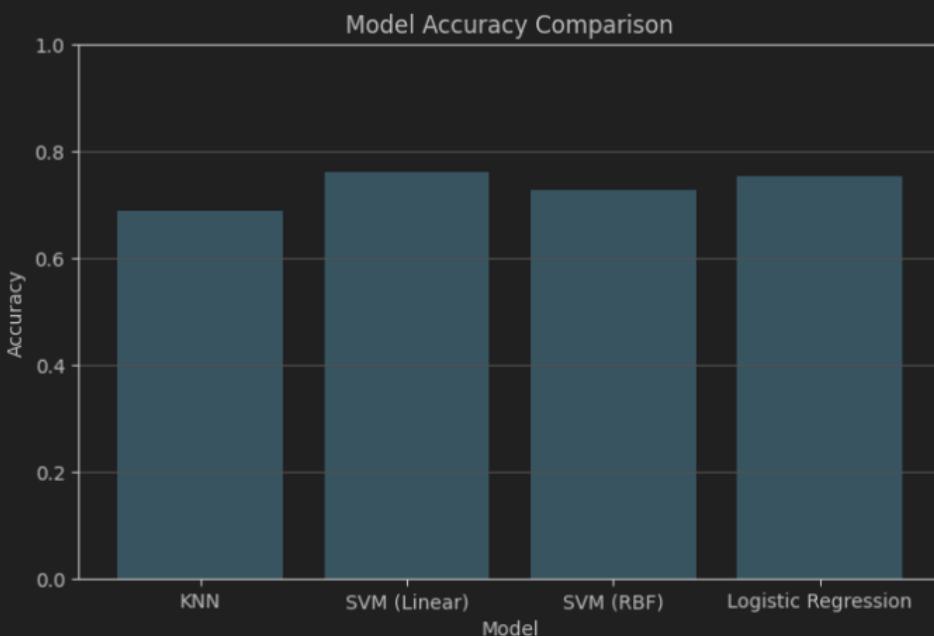
Executed at 2025.03.13 19:57:35 in 7ms

```

plt.figure(figsize=(8, 5))
plt.bar(models, accuracies, color='skyblue')
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.grid(axis='y')
plt.show()

```

Executed at 2025.03.13 19:57:35 in 154ms



Deep Learning > 5 Layers

V1

```
import pandas as pd
import numpy as np
import tensorflow as tf
from keras.api.models import Sequential
from keras.api.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib

Executed at 2025.03.13 19:38:27 in 15s 72ms
# Load dataset  You, 5 minutes ago * Uncommitted changes
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",
           "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
df = pd.read_csv(url, names=columns)

print(df.head())

Executed at 2025.03.13 19:40:32 in 1s 905ms
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin    BMI \
0            6       148             72            35      0  33.6
1            1        85             66            29      0  26.6
2            8       183             64            0      0  23.3
3            1        89             66            23     94  28.1
4            0       137             40            35     168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0            0.627      50       1
1            0.351      31       0
2            0.672      32       1
3            0.167      21       0
4            2.288      33       1

# Split features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

print(X.head())
print(y.head())

Executed at 2025.03.13 19:40:34 in 10ms
```

```

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

Executed at 2025.03.13 19:38:30 in 20ms

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

Executed at 2025.03.13 19:38:30 in 11ms

# Build deep learning model
def create_model():
    model = Sequential([
        Dense(128, activation="relu", input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(64, activation="relu"),
        Dropout(0.3),
        Dense(32, activation="relu"),
        Dense(16, activation="relu"),
        Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
    return model

Executed at 2025.03.13 19:38:30 in 14ms

# Train model
model = create_model()

# Early stopping callback (optional)
from keras.api.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

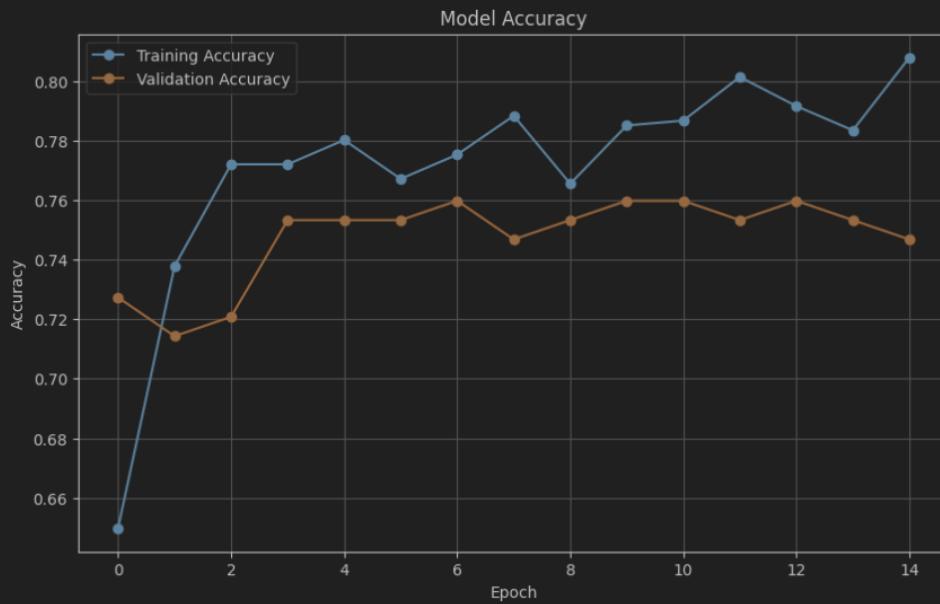
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_data=(X_test, y_test), callbacks=[early_stopping])

Executed at 2025.03.13 19:47:52 in 5s 132ms
Epoch 1/100
D:\Workspace\rb\sample-tracker-api\venv\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/100
62/62 ━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7560 - loss: 0.4795 - val_accuracy: 0.7597 - val_loss: 0.5150
Epoch 8/100
62/62 ━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8050 - loss: 0.4563 - val_accuracy: 0.7468 - val_loss: 0.5116
Epoch 9/100
62/62 ━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7812 - loss: 0.4275 - val_accuracy: 0.7532 - val_loss: 0.5249
Epoch 10/100
62/62 ━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7921 - loss: 0.4525 - val_accuracy: 0.7597 - val_loss: 0.5286
Epoch 11/100
62/62 ━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8144 - loss: 0.4309 - val_accuracy: 0.7597 - val_loss: 0.5328
Epoch 12/100

```

```
1 import matplotlib.pyplot as plt
2
3
4 plt.figure(figsize=(10, 6))
5 plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
6 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
7 plt.title('Model Accuracy')
8 plt.xlabel('Epoch')
9 plt.ylabel('Accuracy')
10 plt.legend()
11 plt.grid(True)
12 plt.show()
```

Executed at 2025.03.13 19:47:53 in 219ms



```
1 # Save model and scaler
2 # model.save("diabetes_model.h5")
3 # joblib.dump(scaler, "scaler.pkl")
4 print("Model and scaler saved successfully!")
```

Executed at 2025.03.13 19:47:53 in 5ms

V2

```

1 # version 2
2 import pandas as pd
3 import numpy as np
4 import tensorflow as tf
5 from keras.api.models import Sequential
6 from keras.api.layers import Dense, Dropout
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import StandardScaler
9 import joblib
Executed at 2025.03.13 19:45:03 in 5s 648ms

1 # Load dataset
2 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv" You, Moments ago · Uncommitt
3 columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin",
4     "BMI", "DiabetesPedigreeFunction", "Age", "Outcome"]
5 df = pd.read_csv(url, names=columns)
6
7 print(df.head())
Executed at 2025.03.13 19:45:05 in 2s 524ms
    Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \
0           6      148          72        35       0  33.6
1           1       85          66        29       0  26.6
2           8      183          64        0       0  23.3
3           1       89          66        23      94  28.1
4           0      137          40        35      168  43.1

   DiabetesPedigreeFunction Age Outcome
0            0.627    50      1
1            0.351    31      0
2            0.672    32      1
3            0.167    21      0
4            2.288    33      1

1 # Split data
2 X = df.drop("Outcome", axis=1)
3 y = df["Outcome"]
4
5 print(X.head())
6 print(y.head())
Executed at 2025.03.13 19:45:05 in 8ms
    Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \
0           6      148          72        35       0  33.6
1           1       85          66        29       0  26.6
2           8      183          64        0       0  23.3
3           1       89          66        23      94  28.1
4           0      137          40        35      168  43.1

   DiabetesPedigreeFunction Age
0            0.627    50
1            0.351    31
2            0.672    32

```

```

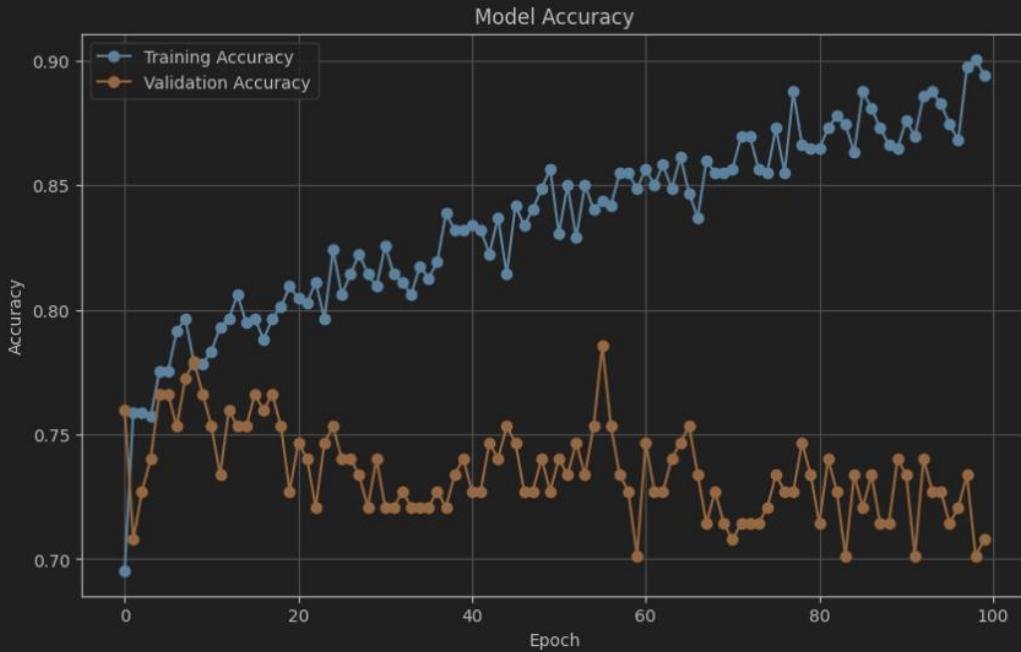
1 # Normalize input features
2 scaler = StandardScaler()
3 X_scaled = scaler.fit_transform(X)
4 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
5                                         # Build deep learning model
6 def create_model():
7     model = Sequential([
8         Dense(128, activation="relu", input_shape=(X_train.shape[1],)),
9         Dropout(0.3),
10        Dense(64, activation="relu"),
11        Dropout(0.3),
12        Dense(32, activation="relu"),
13        Dense(16, activation="relu"),
14        Dense(1, activation="sigmoid")
15    ])
16    model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
17    return model
18
19 Executed at 2025.03.13 19:45:05 in 21ms
20
21 Add Code Cell | Add Markdown Cell
22
23 # Train model
24 model = create_model()  You, 12 minutes ago · Uncommitted changes
25 history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_data=(X_test, y_test))
26
27 Executed at 2025.03.13 19:49:06 in 21s 242ms
28
29 Epoch 1/100
30
31 D:\Workspace\rb\sample-tracker-api\venv\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
32   'input_shape' / 'input_dim' argument to a layer. When using Sequential models, prefer using an 'Input(shape)' object as the first layer in the
33   model instead.
34   super().__init__(activity_regularizer=activity_regularizer, **kwargs)
35
36 Epoch 97/100
37 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8739 - loss: 0.2861 - val_accuracy: 0.7013 - val_loss: 0.7683
38 Epoch 98/100
39 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8919 - loss: 0.2419 - val_accuracy: 0.7403 - val_loss: 0.7671
40 Epoch 94/100
41 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8909 - loss: 0.2796 - val_accuracy: 0.7273 - val_loss: 0.7661
42 Epoch 95/100
43 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8872 - loss: 0.2847 - val_accuracy: 0.7273 - val_loss: 0.7933
44 Epoch 96/100
45 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8771 - loss: 0.2769 - val_accuracy: 0.7143 - val_loss: 0.7850
46 Epoch 97/100
47 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8880 - loss: 0.2749 - val_accuracy: 0.7208 - val_loss: 0.7768
48 Epoch 98/100
49 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8941 - loss: 0.2560 - val_accuracy: 0.7338 - val_loss: 0.8343
50 Epoch 99/100
51 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.9338 - loss: 0.2111 - val_accuracy: 0.7013 - val_loss: 0.8894
52 Epoch 100/100
53 62/62 ━━━━━━━━ 0s 2ms/step - accuracy: 0.8906 - loss: 0.2784 - val_accuracy: 0.7078 - val_loss: 0.8322
54
55
56 import matplotlib.pyplot as plt
57
58
59 plt.figure(figsize=(10, 6))
60 plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')

```

```

1 import matplotlib.pyplot as plt
2
3
4 plt.figure(figsize=(10, 6))
5 plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
6 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
7 plt.title('Model Accuracy')
8 plt.xlabel('Epoch')
9 plt.ylabel('Accuracy')
10 plt.legend()
11 plt.grid(True)
12 plt.show()
13
```

Executed at 2025.03.13 19:49:06 in 533ms



```

1 # Save model & scaler
2 # model.save("diabetes_model.h5")
3 # joblib.dump(scaler, "scaler.pkl")
4 print("Model and scaler saved successfully!")

```

Executed at 2025.03.13 19:49:06 in 5ms

Model and scaler saved successfully!

V3

```
import tensorflow as tf
from keras.api.models import Sequential
from keras.api.layers import Dense, Dropout, BatchNormalization
from keras.api.optimizers import AdamW
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
Executed at 2025.03.13 19:51:04 in 4ms

# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI",
           "DiabetesPedigreeFunction", "Age", "Outcome"]
df = pd.read_csv(url, names=columns)
Executed at 2025.03.13 19:51:06 in 1s 949ms

# Split data
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
Executed at 2025.03.13 19:51:06 in 9ms

# Normalize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
Executed at 2025.03.13 19:51:07 in 53ms

# Build improved deep learning model
def create_model():
    model = Sequential([
        Dense(128, input_shape=(X_train.shape[1],)),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),
        Dropout(0.4),
        Dense(64),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),
        Dropout(0.3),
        Dense(32),
        BatchNormalization(),
        tf.keras.layers.LeakyReLU(alpha=0.1),
        Dense(16, activation="relu"),
        Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer=AdamW(learning_rate=0.001), loss="binary_crossentropy",
                  metrics=["accuracy"])
    return model
Executed at 2025.03.13 19:51:07 in 14ms
```

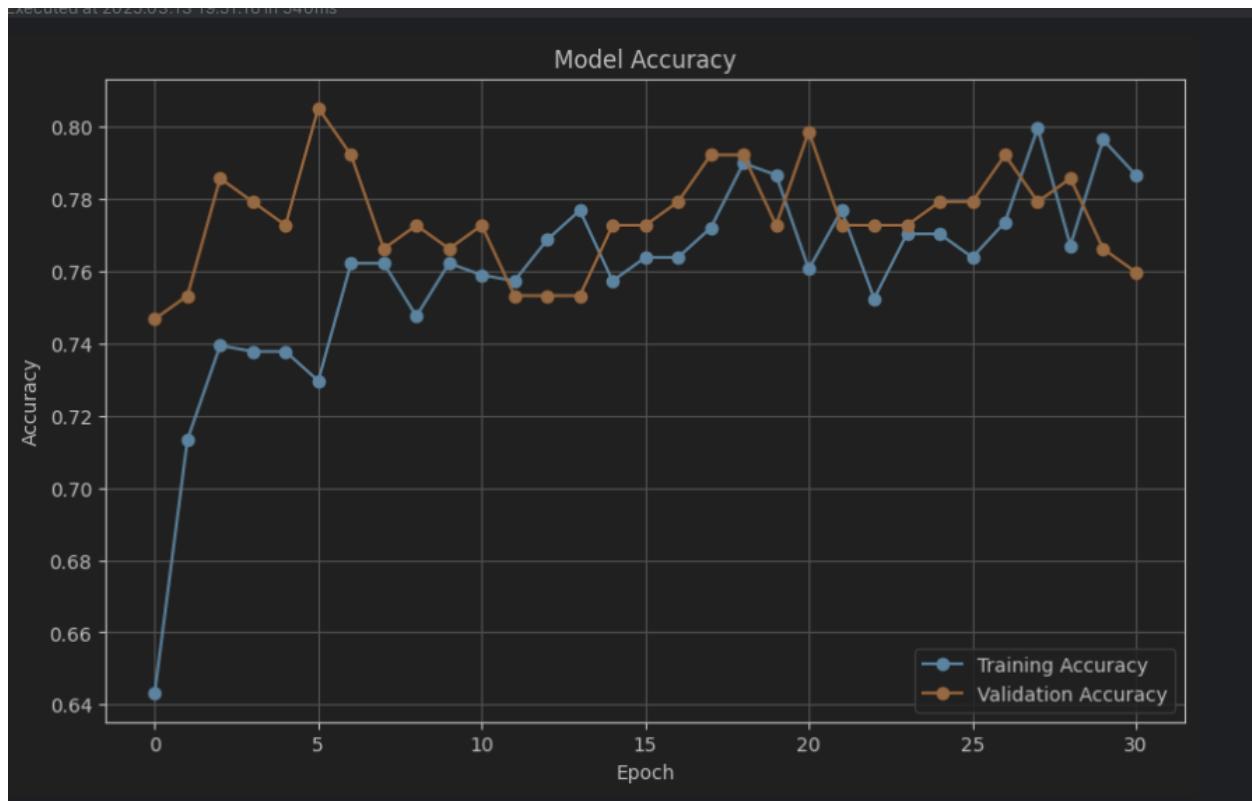
```

1 # Train the model - You, 8 minutes ago * Uncommitted changes
2 model = create_model()
3 early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=10,
4                                                 restore_best_weights=True)
5 history = model.fit(X_train, y_train, epochs=100, validation_data=(X_test, y_test), batch_size=16,
6 callbacks=[early_stopping])
Executed at 2025.03.13 19:51:15 in 8s 567ms

D:\Workspace\rb\sample-tracker-api\venv\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first
model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
D:\Workspace\rb\sample-tracker-api\venv\Lib\site-packages\keras\src\layers\activations\leaky_relu.py:41: UserWarning: Argument
`deprecated`. Use `negative_slope` instead.
    warnings.warn(
Epoch 25/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7622 - loss: 0.4703 - val_accuracy: 0.7727 - val_loss: 0.4981
Epoch 24/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7725 - loss: 0.5041 - val_accuracy: 0.7727 - val_loss: 0.4932
Epoch 25/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7813 - loss: 0.4597 - val_accuracy: 0.7792 - val_loss: 0.4922
Epoch 26/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7443 - loss: 0.4848 - val_accuracy: 0.7792 - val_loss: 0.4884
Epoch 27/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7834 - loss: 0.4334 - val_accuracy: 0.7922 - val_loss: 0.4947
Epoch 28/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7829 - loss: 0.4807 - val_accuracy: 0.7792 - val_loss: 0.4951
Epoch 29/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7644 - loss: 0.4771 - val_accuracy: 0.7857 - val_loss: 0.4920
Epoch 30/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7852 - loss: 0.4612 - val_accuracy: 0.7662 - val_loss: 0.5033
Epoch 31/100
39/39 ━━━━━━━━ 0s 3ms/step - accuracy: 0.7921 - loss: 0.4351 - val_accuracy: 0.7597 - val_loss: 0.4924

1 import matplotlib.pyplot as plt
2
3
4 plt.figure(figsize=(10, 6))
5 plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
6 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
7 plt.title('Model Accuracy')
8 plt.xlabel('Epoch')
9 plt.ylabel('Accuracy')
10 plt.legend()
11 plt.grid(True)
12 plt.show()

```



3. Deploy the models and use interface to allow users entering features, select model for showing the results

```
from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf
app = Flask(__name__)
# Load trained model
model = tf.keras.models.load_model("diabetes_model.h5")
2 usages (2 dynamic)  ↳ vuxhieues
@app.route( rule: "/predict", methods=["POST"])
def predict():
    data = request.json
    features = np.array(data["features"]).reshape(1, -1)
    prediction = model.predict(features)[0][0]
    result = "Diabetic" if prediction > 0.5 else "Non-Diabetic"

    return jsonify({"prediction": result, "probability": float(prediction)})
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)  ↳ vuxhieues, Today + 1
```

```
* Serving Flask app 'server.py'
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [13/Mar/2025 20:04:18] "GET / HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [13/Mar/2025 20:04:18] "GET / HTTP/1.1" 404 -
INFO:werkzeug:127.0.0.1 - - [13/Mar/2025 20:04:18] "GET /favicon.ico HTTP/1.1" 404 -
|
```

```
(venv) PS C:\Users\PC\PycharmProjects\Ass3> cd .\diabetes-frontend\
(venv) PS C:\Users\PC\PycharmProjects\Ass3\diabetes-frontend> node server.js
Frontend server running at http://localhost:2311
```

```
const express : e | () => core.Express = require("express");
const cors : function(any): function(any, a... | {...} = require("cors");
const bodyParser : {...} | {...} = require("body-parser");
const app : any | Express = express();
app.use(cors());
app.use(bodyParser.json());
app.use(express.static( root: "public")); // Serve frontend files
const PORT : number = 2311;
// vuxhieues
app.listen(PORT, hostname: () : void => {
  console.log(`Frontend server running at http://localhost:${PORT}`);
});
```

Pham Duc Chinh

B21DCCN181

Diabetes Prediction

Pregnancies:

12

Glucose:

1200

Blood Pressure:

12

Skin Thickness:

12

Insulin:

120

BMI:

12

Diabetes Pedigree Function:

12

Age:

22

Predict

Prediction: diabetic

Write 5 pages of deep learning

Deep Learning

1. Introduction to Deep Learning

What is Deep Learning?

Deep Learning is a subset of **Machine Learning (ML)** that focuses on using **artificial neural networks (ANN)** to model and solve complex problems. Unlike traditional ML methods that rely heavily on manual feature extraction, deep learning models can automatically discover and learn patterns from data through multi-layered neural networks.

Deep learning models are inspired by the structure and function of the human brain, where neurons process and transmit information. The key idea is that through multiple layers of computation, the model can extract high-level features from raw data, such as:

- Identifying objects in an image
- Understanding speech patterns
- Generating human-like text

Why Deep Learning Matters

Traditional ML methods (e.g., decision trees, SVMs) struggle with large-scale, high-dimensional data. Deep learning models, due to their ability to learn hierarchical representations, have shown significant improvements in:

- Image and video recognition
- Natural language processing (NLP)
- Speech recognition
- Autonomous systems (e.g., self-driving cars)

Key Components of Deep Learning:

1. **Neurons** – Inspired by biological neurons, these are the basic computational units.
2. **Weights and Biases** – Parameters adjusted during training to minimize error.
3. **Activation Functions** – Introduce non-linearity to the network (e.g., ReLU, Sigmoid).
4. **Layers** – Stacked layers of neurons to extract patterns:

- **Input Layer** – Takes raw data as input.
- **Hidden Layers** – Extracts hierarchical features.
- **Output Layer** – Produces final predictions.

Evolution of Deep Learning

Year Milestone

1943	First mathematical model of neurons (McCulloch & Pitts)
1958	Perceptron model (Frank Rosenblatt)
1980s	Backpropagation algorithm rediscovered
2006	Deep Belief Networks (Hinton)
2012	AlexNet wins ImageNet competition, marking DL's breakthrough
2015+	Explosion in AI applications (e.g., GPT, AlphaGo)

2. Neural Network Architectures

2.1. Artificial Neural Networks (ANN)

An **Artificial Neural Network (ANN)** consists of interconnected layers of neurons. Each neuron processes weighted inputs and passes them through an activation function to the next layer.

Structure of an ANN:

- **Input Layer:** Takes input features (e.g., pixels in an image).
- **Hidden Layers:** Perform complex transformations using weighted sums and activation functions.
- **Output Layer:** Outputs the final prediction or classification.

Mathematical Formulation: For a single neuron:

$$y = f(\sum_{i=1}^n w_i x_i + b) \quad y = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

where:

- y = output
- w_i = weights

- x_{ix_xi} = input features
- bbb = bias
- fff = activation function

2.2. Convolutional Neural Networks (CNN)

CNNs are designed for **image and video processing**. They apply convolutional filters to detect patterns in local regions.

Key Components:

- **Convolutional Layer** – Applies filters to detect edges, textures, and patterns.
- **Pooling Layer** – Reduces dimensionality while preserving important features.
- **Fully Connected Layer** – Combines extracted features to produce the output.

 **Example:** CNNs are used in **image classification** (e.g., recognizing cats and dogs).

2.3. Recurrent Neural Networks (RNN)

RNNs are designed for **sequential data** (e.g., text, speech). They have loops that allow them to maintain information over time steps.

Key Components:

- **Hidden State** – Stores information from previous time steps.
- **Vanishing Gradient Problem** – RNNs struggle with long-term dependencies.

 **Solution:** Use **LSTM (Long Short-Term Memory)** or **GRU (Gated Recurrent Unit)** networks to capture long-term dependencies.

 **Example:** RNNs are used in **language translation** and **speech recognition**.

2.4. Transformers

Transformers are state-of-the-art models for handling sequential data without recurrence. They rely on **self-attention** mechanisms to understand context.

Key Components:

- **Self-Attention:** Weighs the importance of different parts of the input sequence.
- **Positional Encoding:** Provides order information to the model.

 **Example:** GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers).

3. Training Deep Learning Models

3.1. Backpropagation

Backpropagation is the key algorithm used to train neural networks by updating weights using the **gradient descent** method.

Steps:

1. **Forward Pass:** Compute the output using current weights.
2. **Loss Calculation:** Measure the error between predicted and actual output.
3. **Backward Pass:** Compute gradients using the chain rule.
4. **Weight Update:** Adjust weights using gradient descent.

3.2. Loss Functions

Loss functions measure the error between predicted and true values:

- **Binary Cross-Entropy:** For binary classification
- **Categorical Cross-Entropy:** For multi-class classification
- **Mean Squared Error (MSE):** For regression problems

3.3. Optimizers

Optimizers control how weights are updated:

- **SGD (Stochastic Gradient Descent):** Updates weights after each sample.
- **Adam (Adaptive Moment Estimation):** Combines momentum and adaptive learning rates.

3.4. Regularization Techniques

Regularization helps prevent overfitting:

- **Dropout:** Randomly disables neurons during training.
 - **L2 Regularization:** Penalizes large weights to prevent complexity.
-

4. Applications of Deep Learning

4.1. Computer Vision

- Image classification (e.g., ResNet)
- Object detection (e.g., YOLO)
- Face recognition (e.g., FaceNet)

4.2. Natural Language Processing (NLP)

- Sentiment analysis
- Machine translation (e.g., Google Translate)
- Chatbots (e.g., GPT)

4.3. Autonomous Systems

- Self-driving cars (e.g., Tesla)
- Robotics (e.g., Boston Dynamics)

4.4. Healthcare

- Disease diagnosis (e.g., cancer detection)
 - Drug discovery
-

5. Challenges and Future of Deep Learning

5.1. Challenges

1. **Data Dependence:** Requires large labeled datasets.
2. **Computational Cost:** Training deep models demands significant GPU/TPU power.
3. **Interpretability:** Difficult to understand how deep models make decisions.
4. **Overfitting:** High complexity increases the risk of overfitting.

5.2. Solutions and Future Directions

1. **Transfer Learning:** Use pre-trained models to reduce data dependence.
2. **Federated Learning:** Train models without sharing data (privacy-focused).
3. **Explainable AI (XAI):** Improve model transparency and interpretability.

-
- 4. **Energy Efficiency:** Develop more efficient architectures (e.g., pruning).
-

6. Conclusion

Deep Learning has revolutionized the field of artificial intelligence by enabling machines to perform complex tasks with human-like accuracy. From image recognition to natural language understanding, deep learning models have demonstrated exceptional performance, surpassing traditional methods. However, challenges like high data requirements and computational costs remain. Future research will focus on improving model efficiency, interpretability, and generalization, making deep learning more accessible and impactful across industries.

 **Deep Learning = Power + Complexity + Innovation** 