KHOA KỸ THUẬT VÀ CÔNG NGHỆ
BỘ MÔN CÔNG NGHỆ THÔNG TIN

CHUYÊN ĐỀ ASP.NET

HỌC KỲ 5, NĂM HỌC 2025

ĐỀ TÀI:

XÂY DỰNG WEBSITE ĐẶT DỊCH VỤ CHĂM SÓC THÚ CƯNG TẠI NHÀ

Giáo viên hướng dẫn:
Họ tên: ThS. Đoàn Phước Miền

Sinh viên thực hiện:
Họ tên: Trần Lâm
MSSV: 170123459
Lớp: DK23TTC11

Trà Vinh, tháng 5 năm 2025

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

Trà Vinh, ngày ..... tháng …… năm ……

Giáo viên hướng dẫn

(Ký tên và ghi rõ họ tên)

NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

Trà Vinh, ngày ..... tháng …… năm ……

Thành viên hội đồng

(Ký tên và ghi rõ họ tên)

LỜI CẢM ƠN

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc đến quý thầy cô trong Bộ môn Công nghệ Thông tin, Trường Đại học Trà Vinh. Trong suốt quá trình học tập, thầy cô không chỉ truyền đạt kiến thức chuyên ngành mà còn tạo điều kiện để em rèn luyện tư duy, kỹ năng và tác phong làm việc chuyên nghiệp.

Em trân trọng gửi lời cảm ơn đến thầy Đoàn Phước Miền – người đã trực tiếp hướng dẫn em trong quá trình thực hiện đồ án tốt nghiệp. Sự hỗ trợ tận tâm, những chỉ dẫn chi tiết cùng những góp ý quý báu từ thầy là nguồn động lực lớn giúp em kiên trì hoàn thành đề tài.

Dù đã nỗ lực hoàn thiện tốt nhất trong khả năng, nhưng với kinh nghiệm thực tế còn hạn chế, bài làm của em vẫn có thể còn một số thiếu sót. Em mong sẽ nhận được những góp ý từ quý thầy cô để có thể tiếp tục cải thiện và phát triển hơn trong chặng đường phía trước.

Em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH - BẢNG BIỂU

TÓM TẮT

Về lý thuyết:

- Kiến trúc 3 lớp (3-tier architecture) trong hệ thống web.

- Công nghệ lập trình ASP.NET Core MVC cho backend và ReactJS cho frontend.

- Entity Framework và mô hình ORM trong quản lý dữ liệu.

- Cơ chế xác thực, phân quyền người dùng với JWT.

- Khái niệm về API, RESTful và giao tiếp giữa client-server.

- Quản lý trạng thái ứng dụng với Redux (React).

- Bảo mật và hiệu suất trong các hệ thống web hiện đại.

Về thực nghiệm:

- Phát triển giao diện người dùng bằng ReactJS kết hợp Material Tailwind.

- Xây dựng backend bằng ASP.NET Core Web API, triển khai các chức năng CRUD và bảo mật.

- Thiết kế và vận hành cơ sở dữ liệu SQL Server.

- Triển khai hệ thống trên nền tảng đám mây: Backend trên Azure Web App, Frontend trên Vercel, và cơ sở dữ liệu trên Azure SQL Server.

- Kiểm thử chức năng người dùng, đặt dịch vụ, xác thực - phân quyền, và hiệu năng hệ thống.

- Ghi nhận kết quả hoạt động ổn định và đúng yêu cầu đề ra.

MỞ ĐẦU

1. Lý do chọn đề tài

- Trong những năm gần đây, số lượng thú cưng được nuôi dưỡng trong các gia đình ngày càng tăng, kéo theo nhu cầu về các dịch vụ chăm sóc thú cưng cũng tăng theo. Các trung tâm chăm sóc thú cưng đang trở thành điểm đến phổ biến cho những người nuôi thú cưng để đảm bảo sức khỏe và phúc lợi cho thú cưng của họ. Tuy nhiên, nhiều trung tâm vẫn gặp khó khăn trong việc quản lý thông tin và dịch vụ một cách hiệu quả.

- Hiện tại, nhiều trung tâm chăm sóc thú cưng sử dụng các phương pháp quản lý thủ công hoặc sử dụng các phần mềm không chuyên dụng, dẫn đến việc quản lý thông tin không nhất quán, mất nhiều thời gian và dễ xảy ra sai sót. Điều này không chỉ ảnh hưởng đến hiệu quả hoạt động của trung tâm mà còn gây ra sự không hài lòng cho khách hàng.

- Vì lý do này, dự án phát triển hệ thống quản lý trung tâm chăm sóc thú cưng được thực hiện nhằm tạo ra một giải pháp quản lý toàn diện, giúp cải thiện hiệu quả hoạt động của các trung tâm chăm sóc thú cưng và nâng cao chất lượng dịch vụ cung cấp cho khách hàng.

2. Mục tiêu nghiên cứu

- Phát triển một hệ thống quản lý trung tâm chăm sóc thú cưng hiện đại, tích hợp đầy đủ các chức năng quản lý thông tin, lịch hẹn, dịch vụ và thanh toán.

- Cung cấp một giao diện người dùng thân thiện và dễ sử dụng, giúp các nhân viên trung tâm dễ dàng quản lý và cập nhật thông tin.

- Đảm bảo tính bảo mật và tính toàn vẹn của dữ liệu, giúp bảo vệ thông tin cá nhân của khách hàng và hồ sơ y tế của thú cưng.

3. Kết quả mong đợi

- Hệ thống quản lý trung tâm chăm sóc thú cưng sẽ hoạt động ổn định và hiệu quả, đáp ứng đầy đủ các yêu cầu chức năng đã đặt ra.

Các lợi ích cụ thể:

- Cải thiện hiệu quả quản lý thông tin và dịch vụ.

- Giảm thiểu thời gian và chi phí quản lý.

- Nâng cao trải nghiệm của khách hàng và thú cưng.

- Triển khai mở rộng sản phẩm trong tương lai

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu gồm:

- Các công nghệ lập trình web: ASP.NET Core, ASP.NET Core MVC, Entity Framework, Razor View, SignalR.

- Công nghệ front-end: React, Material Tailwind.

- Các thao tác CRUD trong môi trường web.

- Cơ sở dữ liệu sử dụng: SQL Server.

Phạm vi nghiên cứu:

- Quản lý thông tin thú cưng: Lưu trữ và quản lý thông tin cá nhân của thú cưng, bao gồm tên, giống, tuổi, và các thông tin y tế liên quan.

- Quản lý lịch hẹn và chăm sóc: Hỗ trợ lên lịch và quản lý các cuộc hẹn khám và chăm sóc thú cưng, đảm bảo không có sự chồng chéo và nhầm lẫn trong lịch trình.

- Quản lý hồ sơ y tế: Lưu trữ và truy xuất hồ sơ y tế của thú cưng, bao gồm các ghi chú y tế, lịch sử điều trị, và các kết quả xét nghiệm.

- Báo cáo và thống kê: Cung cấp các báo cáo và thống kê liên quan đến hoạt động của trung tâm, giúp quản lý theo dõi và đánh giá hiệu quả hoạt động.

CHƯƠNG 1 TỔNG QUAN

1.1 Tổng quan về nền tảng phần mềm mở

1.1.1 Khái niệm nền tảng mở

Hệ sinh thái phần mềm mở đề cập đến các ứng dụng mà mã nguồn được công bố công khai, cho phép cá nhân hoặc tổ chức tự do sử dụng, chỉnh sửa và chia sẻ. Không phụ thuộc vào giấy phép thương mại, loại phần mềm này khuyến khích sự sáng tạo, minh bạch và cộng tác toàn cầu.

1.1.2 Tại sao chọn mã nguồn mở?

- Chi phí đầu tư thấp: Không yêu cầu mua bản quyền.

- Tùy biến sâu: Người dùng có toàn quyền điều chỉnh chức năng theo nhu cầu.

- Tương tác cộng đồng cao: Dễ tiếp cận tài liệu, ví dụ, và sự hỗ trợ từ các nhà phát triển trên toàn thế giới.

- Phát triển liên tục: Cập nhật thường xuyên từ cộng đồng hoặc tổ chức phát hành.

1.1.3 Hành trình hình thành

- Khởi đầu: Năm 1984, với Dự án GNU do Richard Stallman dẫn đầu.

- Bước ngoặt: Sự ra đời của Linux kernel (1991) và việc Netscape công khai mã nguồn (1998) thúc đẩy sự lan rộng khái niệm “open source”.

- Chuẩn hóa: Sự xuất hiện của tổ chức Open Source Initiative (OSI) góp phần định hình các quy tắc pháp lý cho phần mềm mở.

1.1.4 Tác động trong thực tế

- Lợi ích dài hạn: Tiết kiệm, minh bạch, bảo mật nhờ kiểm chứng cộng đồng.

- Tích hợp mạnh mẽ: Dễ dàng nối kết với các hệ thống và nền tảng khác.

- Phù hợp giáo dục và R&D: Là lựa chọn tối ưu cho các tổ chức nghiên cứu, đào tạo hoặc startup.

1.1.5 Hạn chế tiềm ẩn

- Thiếu hỗ trợ chính thức: Chủ yếu phụ thuộc cộng đồng.

- Đòi hỏi kỹ năng: Người dùng cần hiểu biết kỹ thuật để khai thác triệt để.

- Khó sử dụng cho người không chuyên: Một số phần mềm chưa tối ưu UX/UI.

1.1.6 Dự án tiêu biểu

- Ubuntu / Linux Mint – Hệ điều hành mã nguồn mở.
- Nginx / Apache – Máy chủ web phổ biến.
- PostgreSQL / MariaDB – Hệ quản trị cơ sở dữ liệu mạnh mẽ.
- Blender – Ứng dụng đồ họa 3D chuyên nghiệp.
- VS Code – Môi trường phát triển linh hoạt.

1.1.7 Kiểu giấy phép phổ biến

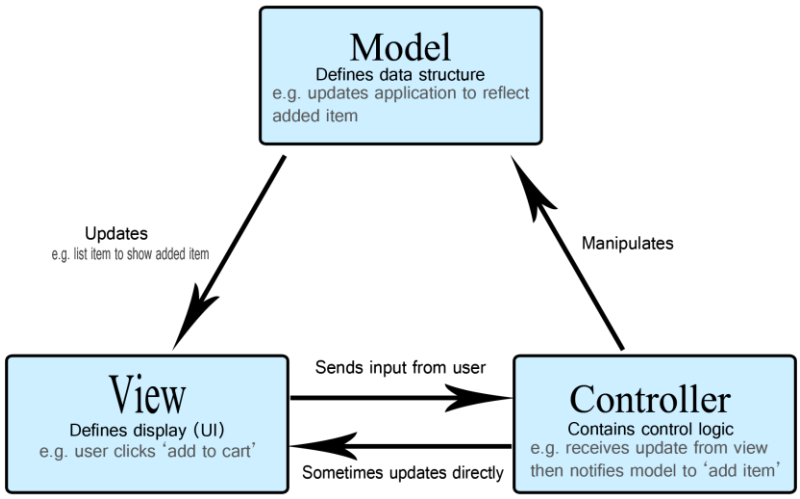
Một số giấy phép phổ biến trong mã nguồn mở:

- GPL (General Public License): Yêu cầu công khai mã nguồn nếu phân phối lại.
- MIT License: Linh hoạt, đơn giản, chỉ cần giữ bản quyền gốc.
- Apache License: Bổ sung điều khoản bằng sáng chế.
- BSD License: Rất thoáng, có thể dùng trong phần mềm đóng.
- Mozilla Public License (MPL): Bán mở, chỉ yêu cầu công khai phần mã đã sửa đổi.

1.2 Mô hình phân tách trong phát triển ứng dụng – Kiến trúc M–V–C

1.2.1 Nguyên lý hoạt động

MVC (Model–View–Controller) là kiến trúc chia tách ứng dụng thành ba phần riêng biệt nhằm tăng khả năng quản lý và mở rộng:



Hình 1.2.1-1 Mô Hình MVC

- M (Model): Xử lý dữ liệu, nghiệp vụ, tương tác với cơ sở dữ liệu.
- V (View): Giao diện người dùng, hiển thị dữ liệu và nhận tương tác.
- C (Controller): Trung gian điều phối, tiếp nhận yêu cầu và xử lý phản hồi.
- Trong ASP.NET Core, Controller gọi Services xử lý logic rồi trả dữ liệu ra View (Razor).

1.2.2 Lợi ích & giới hạn

Ưu điểm:

- Quản lý mã hiệu quả, chia lớp rõ ràng.
- Dễ dàng mở rộng, bảo trì và viết test.
- Thích hợp cho ứng dụng lớn, có nhiều nhóm phát triển.

Nhược điểm:

- Thiết lập phức tạp nếu dự án nhỏ.
- Người mới cần thời gian để hiểu luồng hoạt động giữa các lớp.
- Việc đồng bộ dữ liệu giữa các lớp có thể phát sinh lỗi nếu không kiểm soát tốt.

CHƯƠNG 2 NGHIÊN CỨU LÝ THUYẾT

2.1 Nền tảng phát triển ứng dụng web hiện đại với ASP.NET Core

2.1.1 Tổng quan công nghệ

ASP.NET Core là một framework mã nguồn mở do Microsoft phát triển, cho phép xây dựng các ứng dụng web, API và dịch vụ nền tảng đa hệ điều hành với hiệu năng cao. Đây là phiên bản cải tiến vượt bậc từ ASP.NET truyền thống, loại bỏ các thành phần cũ kỹ, thay vào đó là mô hình kiến trúc nhẹ, module hóa và dễ bảo trì hơn [1].

Hình 2.1.1-1 ASP.NET

2.1.2 Cấu trúc và hệ sinh thái

ASP.NET Core được xây dựng theo hướng modular và hoạt động xoay quanh các thành phần như:

- Web API / MVC / Razor Pages cho các mô hình dự án phổ biến.
- Entity Framework Core để giao tiếp cơ sở dữ liệu dưới dạng ORM.
- Dependency Injection tích hợp sẵn.
- Middleware Pipeline cho phép xử lý tuần tự các request.

2.1.3 Ưu điểm nổi bật

- Đa nền tảng: Chạy tốt trên Windows, Linux, macOS, Azure, Docker,...

- Bảo mật cao: Hỗ trợ HTTPS, xác thực JWT, anti-forgery token.

- Hiệu năng tốt: Tối ưu hóa nhờ Kestrel Server và cơ chế cache hiệu quả.

- Thân thiện với CI/CD: Dễ triển khai thông qua GitHub Actions hoặc Azure DevOps [2].

2.1.4 Một số giới hạn cần cân nhắc

- Cần kiến thức nền tảng .NET để tiếp cận hiệu quả.

- Cấu hình phức tạp nếu không nắm vững cấu trúc DI, Middleware.

- Hosting yêu cầu môi trường .NET Runtime, không phổ biến bằng PHP ở mức shared hosting [3].

2.1.5 Ứng dụng thực tế trong đồ án

Trong hệ thống quản lý trung tâm chăm sóc thú cưng:

- ASP.NET Core đóng vai trò API backend xử lý logic nghiệp vụ.

- Kết hợp JWT Authentication để phân quyền người dùng.

- Sử dụng Entity Framework Core tương tác với SQL Server.

- Triển khai backend trực tiếp trên Azure Web App [4].

2.2 Các công cụ và công nghệ bổ trợ

Công nghệ Back-end:

-.NET: Sử dụng .NET Framework để phát triển các dịch vụ web và API cho hệ thống.

Cơ sở dữ liệu:

- SQL Server: Sử dụng Microsoft SQL Server để lưu trữ và quản lý dữ liệu của hệ thống.

Công nghệ Front-end:

React: Sử dụng React để phát triển giao diện người dùng thân thiện và tương tác.

Công cụ và framework hỗ trợ khác:

- Material Tailwind: Sử dụng bộ giao diện từ Material Tailwind để thiết kế một giao diện người dùng đẹp mắt, dễ sử dụng.

- Entity Framework: Sử dụng Entity Framework để tương tác với cơ sở dữ liệu SQL Server.

- ASP.NET Core: Sử dụng ASP.NET Core để phát triển các dịch vụ back-end.

CHƯƠNG 3 ĐÁNH GIÁ KẾT QUẢ

3.1 Mô tả bài toán

Trong bối cảnh nhu cầu chăm sóc thú cưng ngày càng tăng, các trung tâm chăm sóc thú cưng đang gặp khó khăn trong việc quản lý thủ công thông tin khách hàng, thú cưng, lịch hẹn, hồ sơ y tế và các dịch vụ liên quan. Phương thức quản lý truyền thống thiếu hiệu quả, dễ xảy ra sai sót, không đảm bảo tính bảo mật và gây khó khăn trong việc mở rộng quy mô hoạt động.

Bài toán đặt ra là cần xây dựng một hệ thống phần mềm quản lý trung tâm chăm sóc thú cưng hiện đại, hỗ trợ đầy đủ các chức năng như:

- Quản lý thông tin người dùng và thú cưng.
- Theo dõi hồ sơ y tế, lịch sử khám chữa bệnh.
- Đặt lịch hẹn, quản lý dịch vụ và đặt phòng.
- Phân quyền người dùng theo vai trò (Admin, User).
- Đảm bảo bảo mật, xác thực, và khả năng mở rộng hệ thống.
- Hệ thống phải được triển khai theo mô hình client-server, sử dụng **ASP.NET Core** cho backend, **React** cho frontend và **SQL Server** làm hệ quản trị cơ sở dữ liệu, đồng thời triển khai trên nền tảng **Azure** và **Vercel** để đáp ứng nhu cầu thực tế.

3.1.1 Yêu cầu chức năng (Functional Requirements)

a) *Yêu cầu chức năng (Functional Requirements*

Mô tả: Hệ thống sẽ lưu trữ và quản lý thông tin cá nhân của thú cưng, bao gồm tên, giống, tuổi, màu sắc, và các thông tin liên quan khác.

Chức năng cụ thể:

- Thêm, sửa, xóa và tìm kiếm thông tin thú cưng.
- Liên kết thông tin thú cưng với thông tin chủ sở hữu.
- Cập nhật thông tin y tế cơ bản của thú cưng.

b) *Quản lý hồ sơ bệnh án của thú cưng*

Mô tả: Hệ thống sẽ lưu trữ và quản lý thông tin về bệnh án của từng thú cưng, bao gồm ngày giờ khám, chẩn đoán, tên bác sĩ, kê đơn thuốc, chế độ ăn, và ngày tái khám dự kiến.

Chức năng cụ thể:

- Tạo và quản lý các hồ sơ bệnh án cho từng thú cưng.
- Gửi thông báo nhắc nhở cho chủ sở hữu trước khi ngày tái khám diễn ra.
- Theo dõi và quản lý bệnh tình của thú cưng

c) *Quản lý dịch vụ và đặt chỗ*

Mô tả: Hệ thống sẽ quản lý danh sách các dịch vụ cung cấp tại trung tâm, theo dõi quá trình sử dụng dịch vụ và hỗ trợ quá trình thanh toán nhanh chóng và thuận tiện.

Chức năng cụ thể:

- Quản lý danh sách các dịch vụ chăm sóc thú cưng.
- Hỗ trợ đặt chỗ cho các dịch vụ.
- Theo dõi và quản lý trạng thái sử dụng dịch vụ của từng thú cưng.

3.1.2 Yêu cầu phi chức năng (Non-functional Requirements)

a) *Hiệu suất*

Mô tả: Hệ thống cần đảm bảo hiệu suất hoạt động cao, đáp ứng nhanh chóng các yêu cầu của người dùng và xử lý dữ liệu một cách hiệu quả.

Yêu cầu cụ thể:

- Thời gian phản hồi của hệ thống không quá 2 giây cho các thao tác chính.
- Hệ thống có khả năng xử lý đồng thời ít nhất 1000 yêu cầu từ người dùng.

b) *Bảo mật*

Mô tả: Hệ thống cần đảm bảo tính bảo mật cao, bảo vệ thông tin cá nhân của khách hàng và hồ sơ y tế của thú cưng.

Yêu cầu cụ thể:

- Sử dụng các phương thức mã hóa dữ liệu để bảo vệ thông tin nhạy cảm.
- Áp dụng các cơ chế xác thực và phân quyền người dùng chặt chẽ.
- Đảm bảo hệ thống tuân thủ các quy định về bảo vệ dữ liệu cá nhân.

c) *Khả năng mở rộng*

Mô tả: Hệ thống cần có khả năng mở rộng để đáp ứng nhu cầu tăng trưởng của trung tâm chăm sóc thú cưng trong tương lai.

Yêu cầu cụ thể:

- Hệ thống được thiết kế theo kiến trúc module để dễ dàng mở rộng và nâng cấp.
- Hỗ trợ tích hợp với các hệ thống khác khi cần thiết.

- Khả năng mở rộng về số lượng người dùng và dữ liệu mà không ảnh hưởng đến hiệu suất hệ thống.

3.2 Môi trường phát triển

Để xây dựng và triển khai hệ thống quản lý trung tâm chăm sóc thú cưng, nhóm sử dụng các công cụ và nền tảng sau trong quá trình phát triển:

- Hệ điều hành: Windows 10/11 64-bit

- IDE: Visual Studio 2022 (phát triển backend với ASP.NET Core), Visual Studio Code (phát triển frontend với React)

- Trình quản lý mã nguồn: Git + GitHub

- Cơ sở dữ liệu: Microsoft SQL Server 2019

- Trình quản lý gói:

+ Backend: NuGet

+ Frontend: npm

- Trình duyệt kiểm thử: Google Chrome

- Nền tảng triển khai:

+ Backend: Azure Web App

+ Database: Azure SQL Server

+ Frontend: Vercel

- Công cụ hỗ trợ khác:

+ Postman (kiểm thử API)

+ Azure Data Studio (quản lý CSDL)

3.3 Thiết kế

3.3.1 Kiến trúc ứng dụng

Kiến trúc ứng dụng của hệ thống bao gồm các thành phần chính sau:

a) *Front-end (React)*

- Components: Tạo các thành phần giao diện người dùng.
- Redux: Quản lý trạng thái ứng dụng.
- Router: Quản lý điều hướng giữa các trang.

b) *Back-end (.NET)*

- Controllers: Xử lý các yêu cầu từ frontend và gọi các dịch vụ cần thiết.

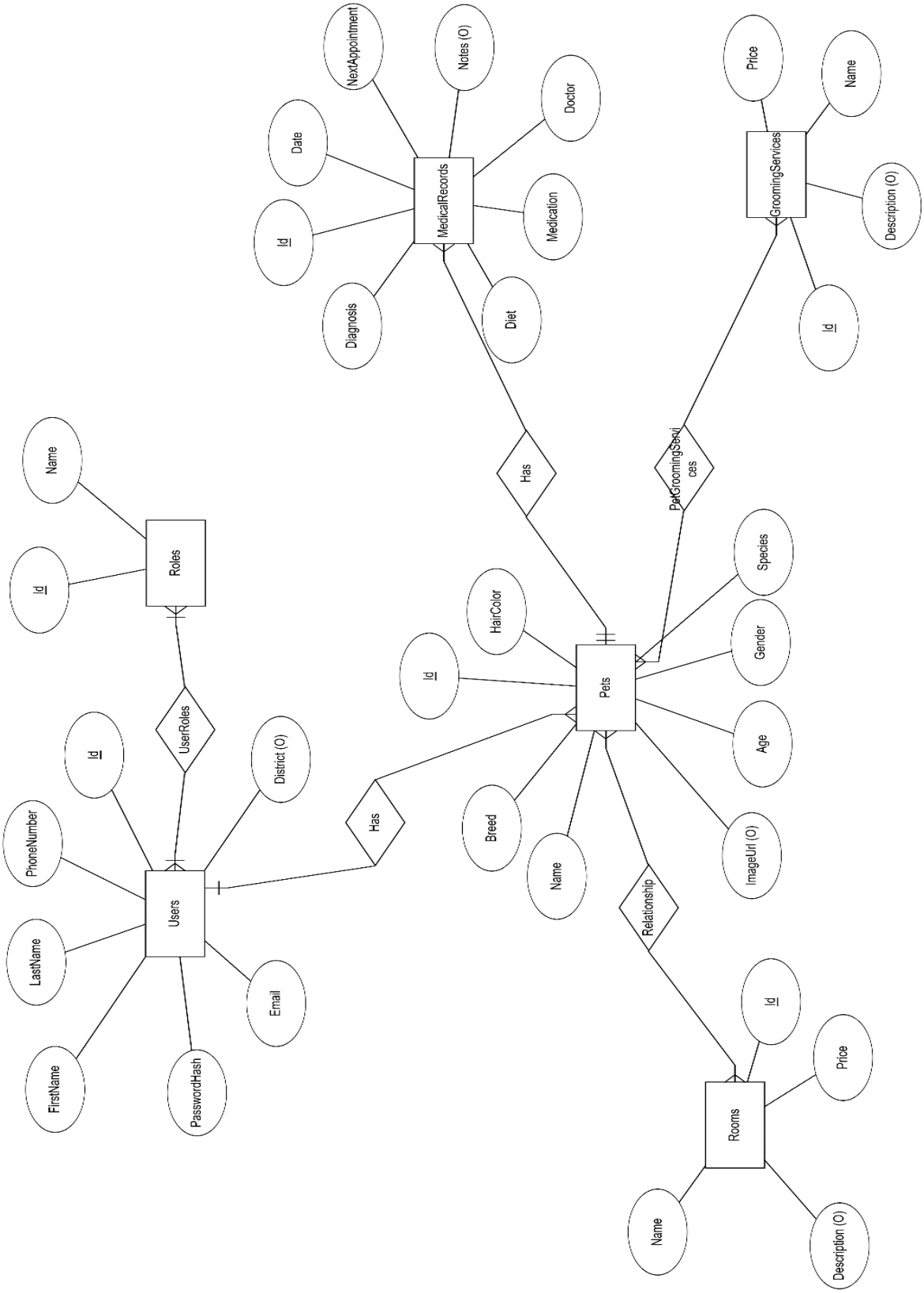
- Services: Chứa logic nghiệp vụ của hệ thống.
- Repositories: Giao tiếp với cơ sở dữ liệu để thực hiện các thao tác CRUD.
- DTOs (Data Transfer Objects): Chuyển đổi dữ liệu giữa các lớp.

c) *Database (SQL Server)*

- Tables: Lưu trữ dữ liệu về thú cưng, lịch hẹn, hồ sơ y tế, dịch vụ, và người dùng.
- Stored Procedures: Thực hiện các thao tác phức tạp với cơ sở dữ liệu.

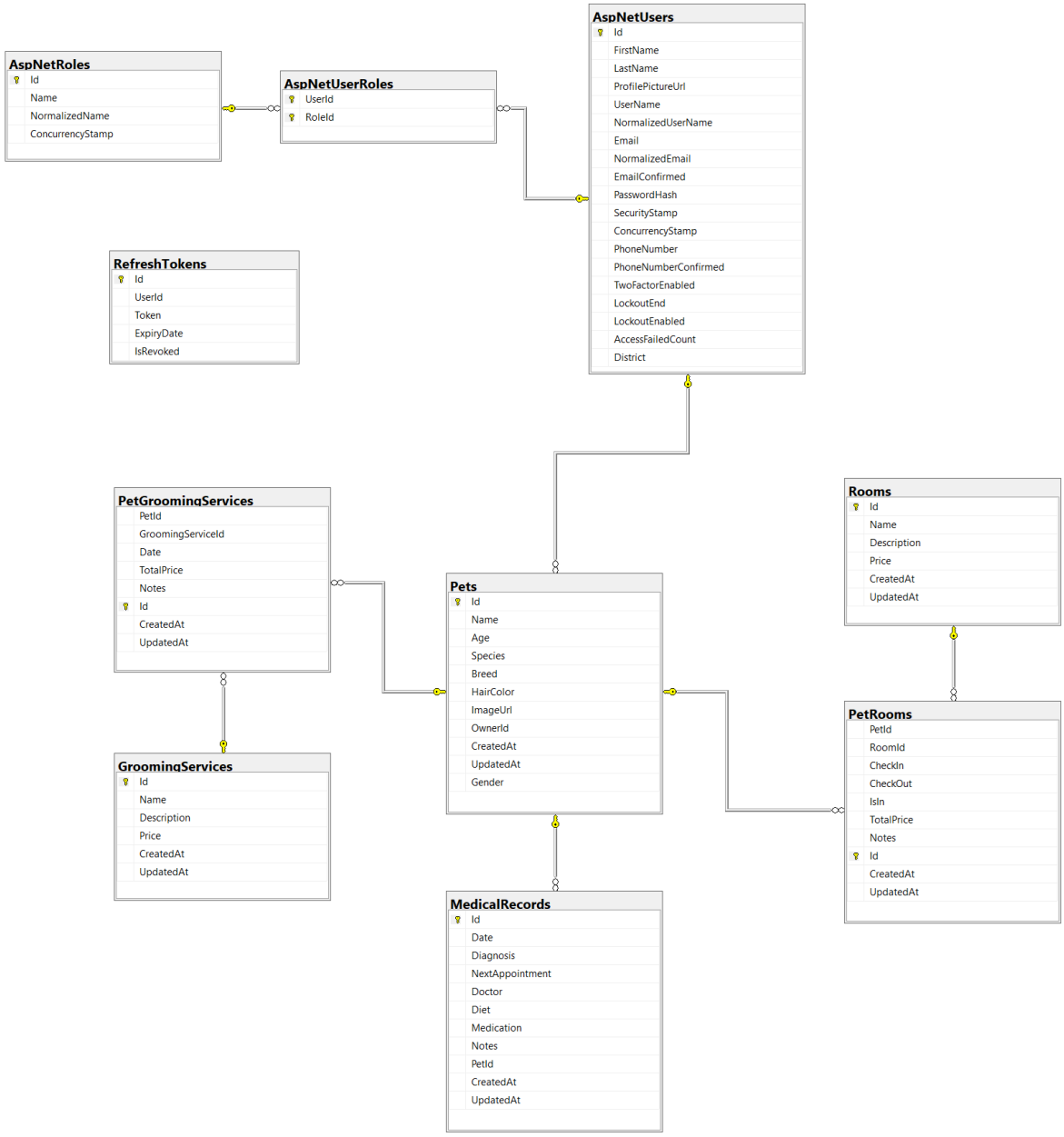
3.3.2 Kiến trúc cơ sở dữ liệu

a) ***Sơ đồ thực thể liên kết***



Hình 3.3.2-1 Sơ đồ thực thể liên kết

b) *Sơ đồ CSDL*



Hình 3.3.2-2 Sơ đồ CSDL

c) *Chi tiết từng bảng*

3.3.2.c.1 AspNetUsers

Mô tả: Lưu trữ thông tin người dùng hệ thống.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**

Id	nvarchar(450)	Khóa chính
FirstName	nvarchar(50)	Tên
LastName	nvarchar(50)	Họ
ProfilePictureUrl	nvarchar(MAX)	Link ảnh đại diện
UserName	nvarchar(256)	Tên người dùng
Email	nvarchar(256)	Email
PasswordHash	nvarchar(256)	Mật khẩu đã ma hoá
PhoneNumber	nvarchar(50)	Số điện thoại
District	nvarchar(50)	Quận/Huyện

3.3.2.c.2 AspNetRoles

Mô tả: Lưu trữ thông tin về các vai trò người dùng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	nvarchar(450)	Khóa chính
Name	nvarchar(256)	Tên role
NormalizedName	nvarchar(256)	

3.3.2.c.3 AspNetUserRoles

Mô tả: Liên kết giữa người dùng và vai trò.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
UserId	nvarchar(450)	Khóa tham chiếu đến AspNetUsers

RoleId	nvarchar(450)	Khoá tham chiếu đến AspNetRoles

3.3.2.c.4 RefreshTokens

Mô tả: Lưu trữ thông tin về các mã làm mới (refresh token) để duy trì phiên đăng nhập.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	int	Khóa chính
UserId	nvarchar(450)	Tên role
Token	nvarchar(256)	Refresh token
ExpiryDate	datetime2(7)	Ngày hết hạn
IsRevoked	Bit	Trạng thái huỷ

3.3.2.c.5 Pets

Mô tả: Lưu trữ thông tin về thú cưng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	int	Khóa chính
Name	nvarchar(50)	Tên thú cưng
Age	int	Tuổi thú cưng
Species	nvarchar(50)	Loài
Breed	nvarchar(50)	Giống
ImageUrl	nvarchar(MAX)	Link ảnh
Gender	Nvarchar(50)	Giới tính

OwnerId	nvarchar(450)	Tham chiếu đến AspNetUsers

3.3.2.c.6 MedicalRecords

Mô tả: Lưu trữ hồ sơ y tế của thú cưng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	int	Khóa chính
Date	datetime2(7)	Ngày ghi nhận hồ sơ
Diagnosis	nvarchar(MAX)	Chẩn đoán
Doctor	nvarchar(256)	Tên bác sĩ
Diet	nvarchar(MAX)	Chế độ ăn
Medication	nvarchar(MAX)	Đơn thuốc
Notes	nvarchar(MAX)	Ghi chú

3.3.2.c.7 GroomingServices

Mô tả: Lưu trữ thông tin về các dịch vụ chăm sóc thú cưng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	int	Khóa chính
Name	nvarchar(256)	Tên dịch vụ
Description	nvarchar(MAX)	Mô tả dịch vụ
Price	decimal(18, 2)	Giá dịch vụ

3.3.2.c.8 PetGroomingServices

Mô tả: Liên kết giữa thú cưng và các dịch vụ chăm sóc đã đặt cho thú cưng đó.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	Int	Khoá chính
PetId	int	Khóa tham chiếu đến Pets
GroomingServiceId	Int	Khoá tham chiếu đến GroomingServices
Date	datetime2(7)	Ngày sử dụng dịch vụ
TotalPrice	decimal(18, 2)	Tổng giá tiền
Notes	nvarchar(MAX)	Ghi chú

3.3.2.c.9 Rooms

Mô tả: Lưu trữ thông tin về các phòng chăm sóc và lưu giữ thú cưng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	int	Khóa chính
Name	nvarchar(256)	Tên phòng
Description	nvarchar(MAX)	Mô tả dịch vụ
Price	decimal(18, 2)	Giá phòng

3.3.2.c.10 PetRooms

Mô tả: Liên kết giữa thú cưng và các phòng đã sử dụng.

Thuộc tính:

Tên	**Kiểu dữ liệu**	**Mô tả**
Id	Int	Khoá chính

PetId	int	Khóa tham chiếu đến Pets
RoomId	Int	Khoá tham chiếu đến Room
CheckIn	datetime2(7)	Ngày nhận phòng
CheckOut	datetime2(7)	Ngày trả phòng
IsIn	Bit	Trạng thái phòng
TotalPrice	decimal(18, 2)	Tổng giá tiền
Notes	nvarchar(MAX)	Ghi chú

3.3.3 Xác định Use case

a) *Biểu đồ Use case*

Dưới đây là biểu đồ Use Case cho hệ thống quản lý trung tâm chăm sóc thú cưng, mô tả các thực thể và chức năng chính của hệ thống.

Hình 3.3.3-1 Sơ đồ use case

b) *Các Use case chính*

3.3.3.b.1 Quản lý tài khoản người dùng

- Đăng nhập/Đăng xuất
- Thêm/Sửa/Xóa tài khoản người dùng
- Phân quyền người dùng

3.3.3.b.2 Quản lý thông tin thú cưng

- Thêm/Sửa/Xóa thông tin thú cưng
- User chỉ có thể truy cập vào thông tin của các thú cưng do mình sở hữu.

3.3.3.b.3 Quản lý hồ sơ bệnh án

- Thêm/Sửa/Xóa hồ sơ y tế
- Tìm kiếm hồ sơ y tế

3.3.3.b.4 Quản lý dịch vụ chăm sóc

- Thêm/Sửa/Xóa dịch vụ chăm sóc
- Quản lý danh sách dịch vụ (quyền Admin)

3.3.3.b.5 Quản lý đặt phòng và đặt dịch vụ

- Đặt phòng cho thú cưng
- Cập nhật thông tin đặt phòng
- Xóa thông tin đặt phòng

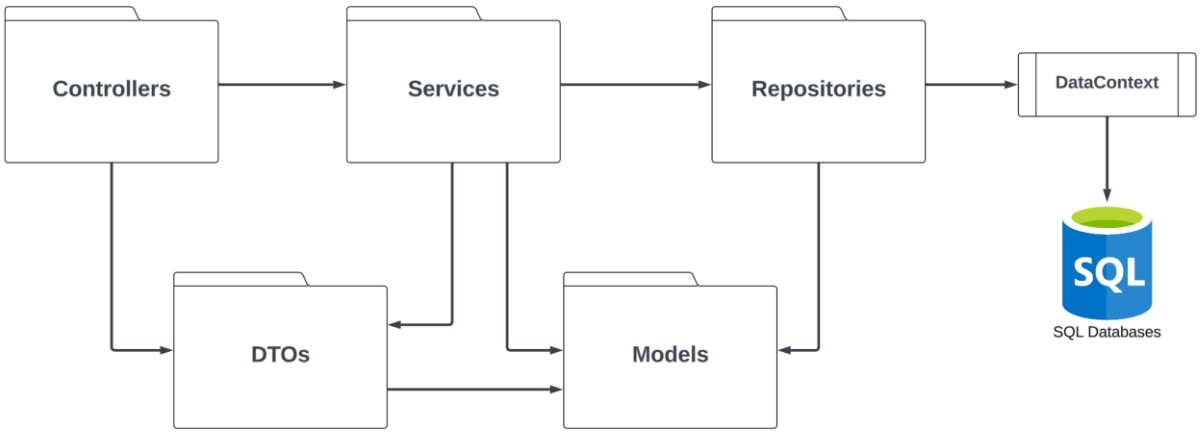
3.3.3.b.6 Quản lý báo cáo và thống kê (quyền Admin)

- Xem báo cáo
- Tạo thống kê

3.3.4 Thiết kế các package

Hệ thống quản lý trung tâm chăm sóc thú cưng được tổ chức theo các package chính để đảm bảo tính modular và dễ bảo trì. Các package chính bao gồm: Controllers, Models, DTOs, Repositories, và Services.

a) *Biểu đồ package tổng quát*



Hình 3.3.4-1 Biểu đồ package tổng quát

b) *Vai trò của từng package*

3.3.4.b.1 Controllers

Nhận các yêu cầu từ người dùng, xử lý các yêu cầu đó bằng cách gọi các service tương ứng và trả về phản hồi (dưới dạng DTO) cho người dùng.

3.3.4.b.2 Models

Biểu diễn cấu trúc dữ liệu trong cơ sở dữ liệu, đại diện cho các thực thể trong hệ thống như User, Pet, MedicalRecord, GroomingService, Room, PetRoom, và PetGroomingService.

3.3.4.b.3 DTOs (Data Transfer Objects)

Bảo vệ dữ liệu: DTOs giúp tách biệt các thực thể cơ sở dữ liệu (Models) khỏi tầng presentation (Controllers). Điều này đảm bảo rằng các chi tiết nội bộ của Models không bị lộ ra ngoài khi dữ liệu được truyền tải qua các giao diện ứng dụng.

Bảo mật: Giúp hạn chế việc truy cập trực tiếp vào các thuộc tính nhạy cảm của Models. Bằng cách chỉ truyền những thông tin cần thiết qua DTOs, hệ thống có thể giảm thiểu nguy cơ lộ thông tin nhạy cảm.

3.3.4.b.4 Repositories

Repositories đóng vai trò như một tầng trung gian giữa các Services và cơ sở dữ liệu, chịu trách nhiệm thực hiện các thao tác CRUD (Create, Read, Update, Delete) với

các thực thể (Entities) trong cơ sở dữ liệu. Các chức năng chính của Repositories bao gồm:

Quản lý truy cập dữ liệu:

- Cung cấp các phương thức để truy xuất và thay đổi dữ liệu từ cơ sở dữ liệu mà không cần lộ chi tiết của các thao tác SQL.

- Đảm bảo tính nhất quán và toàn vẹn của dữ liệu khi thực hiện các thao tác với cơ sở dữ liệu.

Tách biệt logic truy xuất dữ liệu:

- Tách biệt logic truy xuất dữ liệu khỏi logic nghiệp vụ, giúp tăng tính tái sử dụng và dễ bảo trì của mã nguồn.

- Giúp các Services tập trung vào logic nghiệp vụ mà không cần quan tâm đến cách dữ liệu được lưu trữ hay truy xuất.

Cung cấp giao diện truy xuất dữ liệu:

- Định nghĩa các giao diện truy xuất dữ liệu chung để có thể dễ dàng thay đổi hoặc mở rộng mà không ảnh hưởng đến các phần khác của hệ thống.

3.3.4.b.5 Services

Services trong hệ thống quản lý trung tâm chăm sóc thú cưng đóng vai trò quan trọng trong việc thực hiện logic nghiệp vụ (business logic) của ứng dụng. Chúng là tầng trung gian giữa Controllers và Repositories, đảm bảo rằng các yêu cầu từ người dùng được xử lý đúng cách và dữ liệu được thao tác hợp lý. Dưới đây là các vai trò chi tiết của Services:

- Xử lý nghiệp vụ: Services là nơi chứa đựng logic nghiệp vụ của ứng dụng. Mọi thao tác tính toán, kiểm tra, và xử lý dữ liệu đều được thực hiện tại đây.

- Truy xuất và lưu trữ dữ liệu: Services sử dụng các Repositories để truy xuất và lưu trữ dữ liệu từ cơ sở dữ liệu. Chúng gọi các phương thức của Repositories để thực hiện các thao tác CRUD (Create, Read, Update, Delete).

- Đóng gói logic truy vấn: Services có thể thực hiện các truy vấn phức tạp hoặc thao tác dữ liệu bằng cách kết hợp nhiều phương thức từ các Repositories khác nhau.

- Giảm phụ thuộc trực tiếp: Bằng cách sử dụng Services, Controllers không cần phải biết chi tiết về cách dữ liệu được lưu trữ hoặc truy xuất, giảm phụ thuộc và tăng tính linh hoạt.

- Chuyển đổi dữ liệu: Services tạo ra và sử dụng DTOs để chuyển dữ liệu giữa các tầng của ứng dụng. Chúng chuyển đổi dữ liệu từ các Models sang DTOs trước khi gửi đến Controllers và ngược lại.

c) *Phụ thuộc và quan hệ giữa các package*

Trong kiến trúc phần mềm, các package phụ thuộc lẫn nhau và tham chiếu lẫn nhau để thực hiện các chức năng của hệ thống. Dưới đây là giải thích về dependency (sự phụ thuộc) và reference (sự tham chiếu) giữa các package chính trong hệ thống quản lý trung tâm chăm sóc thú cưng.

3.3.4.c.1 Controllers

Services: Controllers phụ thuộc vào Services để thực hiện các logic nghiệp vụ. Mỗi khi có một yêu cầu từ người dùng, controller sẽ gọi các phương thức trong các service tương ứng để xử lý yêu cầu đó.

Ví dụ: UserController sẽ gọi UserService để thực hiện các thao tác liên quan đến người dùng.

DTOs: Controllers sử dụng DTOs để nhận dữ liệu từ người dùng và trả dữ liệu về cho người dùng. DTOs giúp chuyển dữ liệu một cách an toàn giữa tầng presentation và tầng business logic.

Ví dụ: PetController sẽ nhận PetDTO từ người dùng để thêm thông tin thú cưng mới và trả PetDTO về cho người dùng sau khi lấy thông tin thú cưng từ PetService.

3.3.4.c.2 Services

Repositories: Services phụ thuộc vào Repositories để thực hiện các thao tác CRUD trên cơ sở dữ liệu. Mỗi service sẽ gọi các phương thức của repository tương ứng để truy xuất hoặc cập nhật dữ liệu trong cơ sở dữ liệu.

Ví dụ: PetService sẽ gọi PetRepository để lưu trữ hoặc lấy thông tin thú cưng từ cơ sở dữ liệu.

DTOs: Services sử dụng DTOs để nhận dữ liệu từ controllers và trả dữ liệu về cho controllers sau khi xử lý logic nghiệp vụ.

Ví dụ: MedicalRecordService sẽ nhận MedicalRecordDTO từ MedicalRecordController, xử lý logic và trả về MedicalRecordDTO đã cập nhật cho người dùng

Models: Services có thể sử dụng Models để biểu diễn dữ liệu dưới dạng đối tượng trong quá trình xử lý logic nghiệp vụ.

Ví dụ: MedicalRecordService sẽ làm việc với đối tượng MedicalRecord trước khi chuyển đổi sang MedicalRecordDTO để trả về cho MedicalRecordController.

3.3.4.c.3 Repositories

Models: Repositories phụ thuộc vào Models để biểu diễn cấu trúc dữ liệu trong cơ sở dữ liệu. Mỗi repository sẽ tương tác với các đối tượng model để thực hiện các thao tác CRUD.

Ví dụ: UserRepository sẽ làm việc với đối tượng User để lưu trữ và truy xuất thông tin người dùng.

DataContext: Repositories phụ thuộc vào một lớp DataContext để quản lý kết nối và giao tiếp với cơ sở dữ liệu.

Ví dụ: PetRepository sẽ sử dụng DbContext để thực hiện các truy vấn SQL.

3.3.4.c.4 DTOs

Models: DTOs thường được ánh xạ từ Models và ngược lại. Chúng được sử dụng để truyền dữ liệu giữa các tầng của ứng dụng mà không làm lộ các chi tiết của mô hình dữ liệu nội bộ.

Ví dụ: PetDTO sẽ ánh xạ từ đối tượng Pet trong PetService trước khi được gửi đến PetController.

3.3.4.c.5 Models

Repositories: Models được sử dụng bởi Repositories để biểu diễn cấu trúc dữ liệu khi tương tác với cơ sở dữ liệu.

Ví dụ: Appointment model sẽ được sử dụng trong AppointmentRepository để thực hiện các thao tác CRUD.

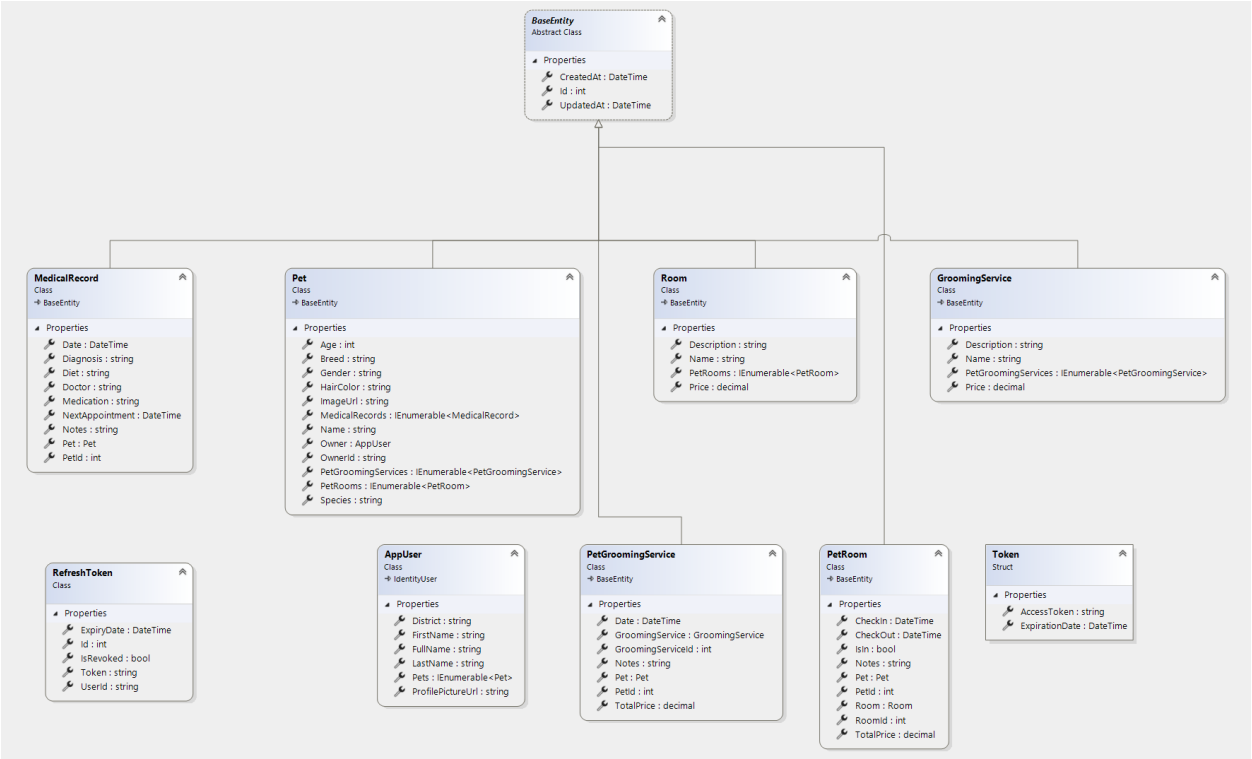
Services: Models cũng có thể được sử dụng trong Services để xử lý logic nghiệp vụ trước khi chuyển đổi sang DTOs.

Ví dụ: MedicalRecord sẽ được sử dụng trong MedicalRecordService để xử lý dữ liệu trước khi chuyển đổi sang MedicalRecordDTO.

3.3.5 Chi tiết các lớp trong các package

a) *Models*

Sơ đồ lớp trong package Models:



Hình 3.3.5-1 Sơ đồ lớp trong package Models

BaseEntity: Lớp cơ bản cung cấp các thuộc tính chung như Id, CreatedAt, và UpdatedAt.

AppUser: Đại diện cho người dùng hệ thống, có mối quan hệ 1-n với Pet.

RefreshToken: Đại diện cho các refresh token được dùng để cấp lại token trong hệ thống xác thực người dùng.

Pet: Đại diện cho thú cưng, có mối quan hệ 1-n với MedicalRecord, PetGroomingService, và PetRoom.

MedicalRecord: Đại diện cho hồ sơ y tế của thú cưng, liên kết một chiều với Pet.

Room: Đại diện cho phòng chăm sóc thú cưng, có mối quan hệ 1-n với PetRoom.

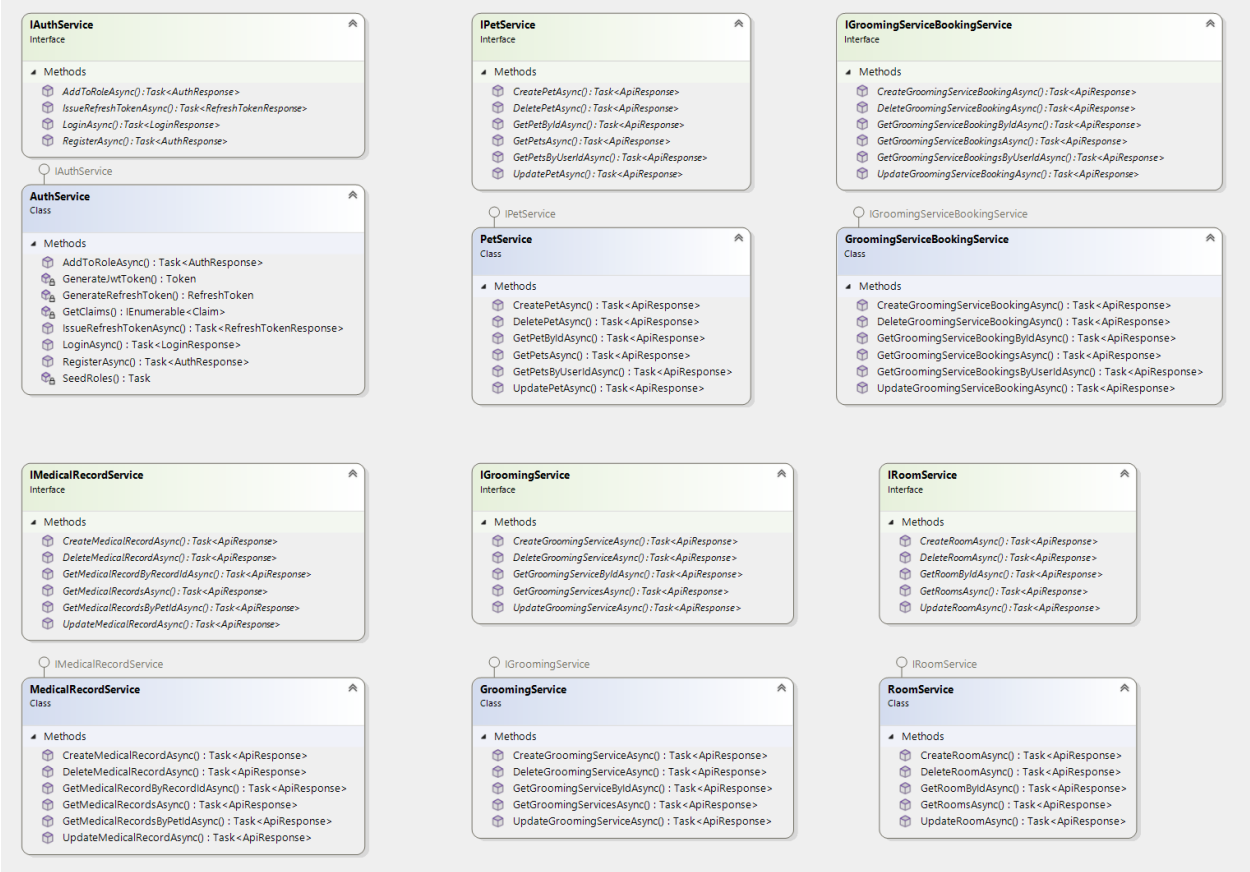
GroomingService: Đại diện cho dịch vụ chăm sóc lông thú cưng, có mối quan hệ 1-n với PetGroomingService.

PetGroomingService: Đại diện cho các dịch vụ chăm sóc lông mà thú cưng đã sử dụng, liên kết hai chiều với Pet và GroomingService.

PetRoom: Đại diện cho việc đặt phòng của thú cưng, liên kết hai chiều với Pet và Room.

b) *Services*

Sơ đồ lớp trong package Services:



Hình 3.3.5-2 Sơ đồ lớp trong package Services

3.3.5.b.1 Cấu trúc chung của các class trong Services

Interfaces (Giao diện)

Mục đích: Các giao diện (interface) định nghĩa các phương thức mà lớp service cần triển khai. Các giao diện này xác định các hành động chính mà service có thể thực hiện, chẳng hạn như tạo, đọc, cập nhật, và xóa (CRUD) các đối tượng liên quan.

Lợi ích: Việc sử dụng interface giúp tách biệt giữa các lớp định nghĩa và các lớp triển khai, cho phép dễ dàng thay đổi hoặc mở rộng các lớp service mà không ảnh hưởng đến các thành phần khác của hệ thống.

Classes (Lớp)

Mục đích: Các lớp service triển khai các interface, thực hiện logic nghiệp vụ và các thao tác cụ thể. Các lớp này chứa các phương thức được định nghĩa trong giao diện và có thể gọi đến các repository để truy xuất hoặc cập nhật dữ liệu trong cơ sở dữ liệu.

Lợi ích: Tách biệt logic nghiệp vụ khỏi tầng controller và repository, giúp duy trì một hệ thống gọn gàng, dễ bảo trì và mở rộng.

3.3.5.b.2 Giao tiếp với các thành phần khác của các package

Giao tiếp với Controllers

Vai trò: Các lớp service nhận yêu cầu từ controllers và thực hiện logic nghiệp vụ tương ứng. Controllers gọi các phương thức của services để xử lý các yêu cầu từ người dùng.

Giao tiếp với Repositories

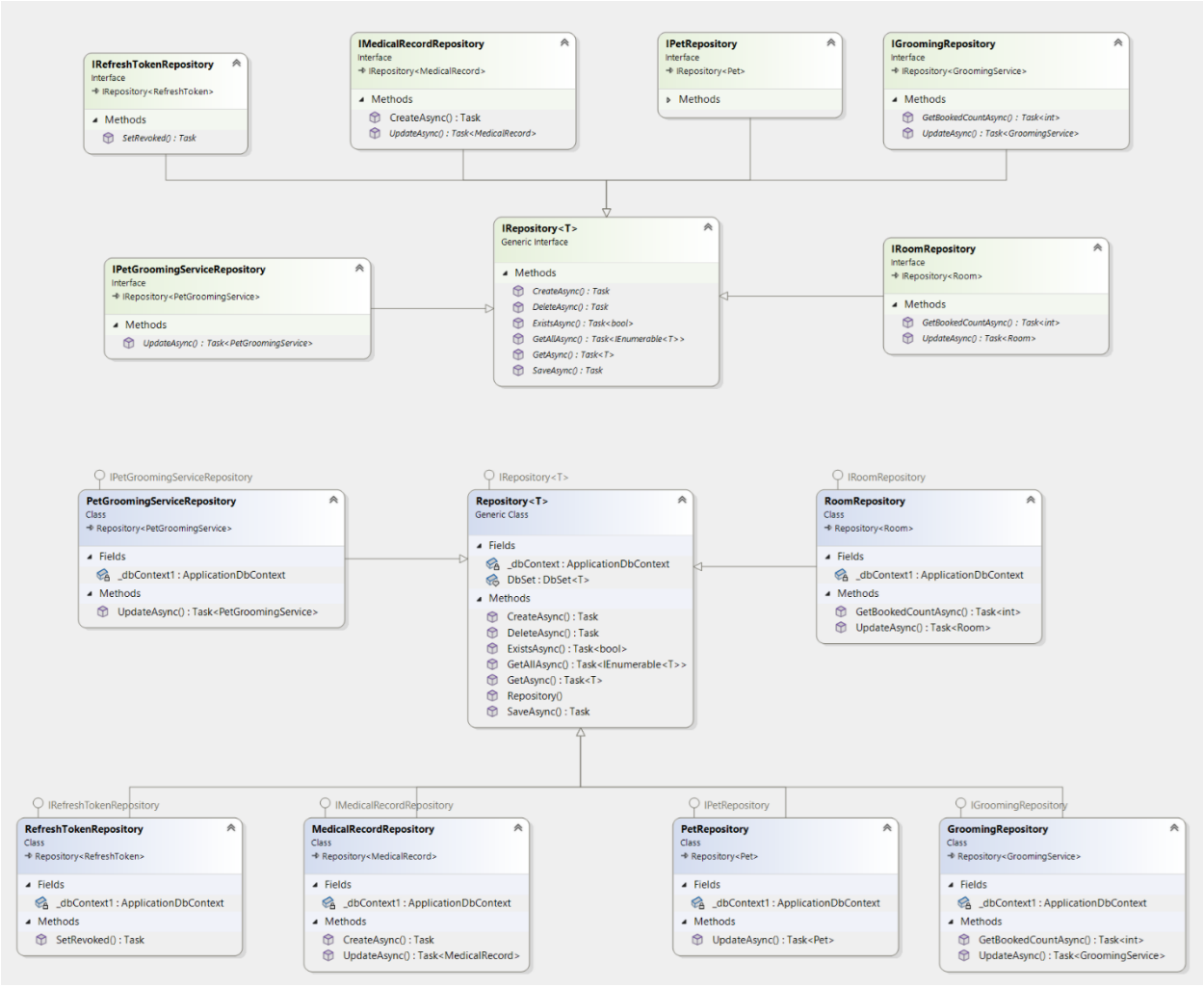
Vai trò: Các lớp service sử dụng các repository để thực hiện các thao tác CRUD với cơ sở dữ liệu. Repository cung cấp các phương thức để truy xuất và thay đổi dữ liệu.

Giao tiếp với Models và DTOs

Vai trò: Các lớp service thường sử dụng các models để biểu diễn dữ liệu nội bộ và DTOs để chuyển đổi dữ liệu giữa các tầng.

c) *Repositories*

Sơ đồ lớp trong package Repositories



Hình 3.3.5-3 Sơ đồ lớp trong package Repositories

3.3.5.c.1 Cấu trúc chung của các class trong Repositories

Interfaces (Giao diện)

Mục đích: Các giao diện (interface) định nghĩa các phương thức mà các lớp repository cần triển khai. Chúng cung cấp một hợp đồng rõ ràng về các thao tác CRUD và các thao tác đặc thù khác mà repository có thể thực hiện.

Lợi ích: Việc sử dụng interface giúp tách biệt định nghĩa và triển khai, cho phép dễ dàng thay đổi hoặc mở rộng các lớp repository mà không ảnh hưởng đến các phần khác của hệ thống.

Generic Repository

Mục đích: Lớp generic repository cung cấp các phương thức CRUD chung cho các thực thể. Nó giúp giảm thiểu việc lặp lại mã nguồn và tăng tính tái sử dụng.

Lợi ích: Cung cấp các thao tác chung cho nhiều loại thực thể khác nhau, giúp giảm thiểu việc lặp lại mã nguồn và tăng hiệu quả phát triển.

Concrete Repositories

Mục đích: Các lớp repository cụ thể triển khai các interface và có thể mở rộng hoặc tùy chỉnh các phương thức chung từ generic repository để phù hợp với nhu cầu cụ thể của từng loại thực thể.

Lợi ích: Đảm bảo rằng mỗi loại thực thể có các thao tác CRUD riêng biệt và có thể thêm các phương thức tùy chỉnh nếu cần. Ví dụ, IRoomRepository và IGroomingService khai báo thêm 1 phương thức so với interface IRepository là GetBookedCount để lấy số lần mà phòng và dịch vụ đã được đặt.

3.3.5.c.2 Quan hệ giữa các class và interface

Quan hệ triển khai

Mỗi lớp repository cụ thể triển khai một hoặc nhiều giao diện tương ứng. Ví dụ, PetRepository triển khai IPetRepository. Điều này đảm bảo rằng lớp repository tuân thủ các hợp đồng được định nghĩa trong giao diện.

Quan hệ kế thừa

Các lớp repository cụ thể có thể kế thừa từ generic repository để sử dụng các phương thức chung. Ví dụ, PetRepository có thể kế thừa từ Repository<T> để sử dụng các phương thức CRUD chung và thêm các phương thức đặc thù cho Pet.

3.3.5.c.3 Giao tiếp với các thành phần của các package khác

Giao tiếp với Services

Vai trò: Các lớp repository cung cấp các phương thức để services truy xuất và thay đổi dữ liệu trong cơ sở dữ liệu. Services gọi các phương thức của repository để thực hiện các thao tác CRUD và các thao tác đặc thù khác.

Quy trình:

- Service gọi các phương thức của repository để thực hiện các thao tác với cơ sở
- dữ liệu.
- Repository truy cập cơ sở dữ liệu và thực hiện các thao tác cần thiết.
- Repository trả kết quả về cho service.
- Service tiếp tục xử lý và trả kết quả cuối cùng về cho controller.

Giao tiếp với Data Context

Vai trò: Các lớp repository sử dụng Data Context để truy cập cơ sở dữ liệu. Data Context quản lý kết nối và giao tiếp với cơ sở dữ liệu, cung cấp các DbSet cho các thực thể khác nhau.

Quy trình:

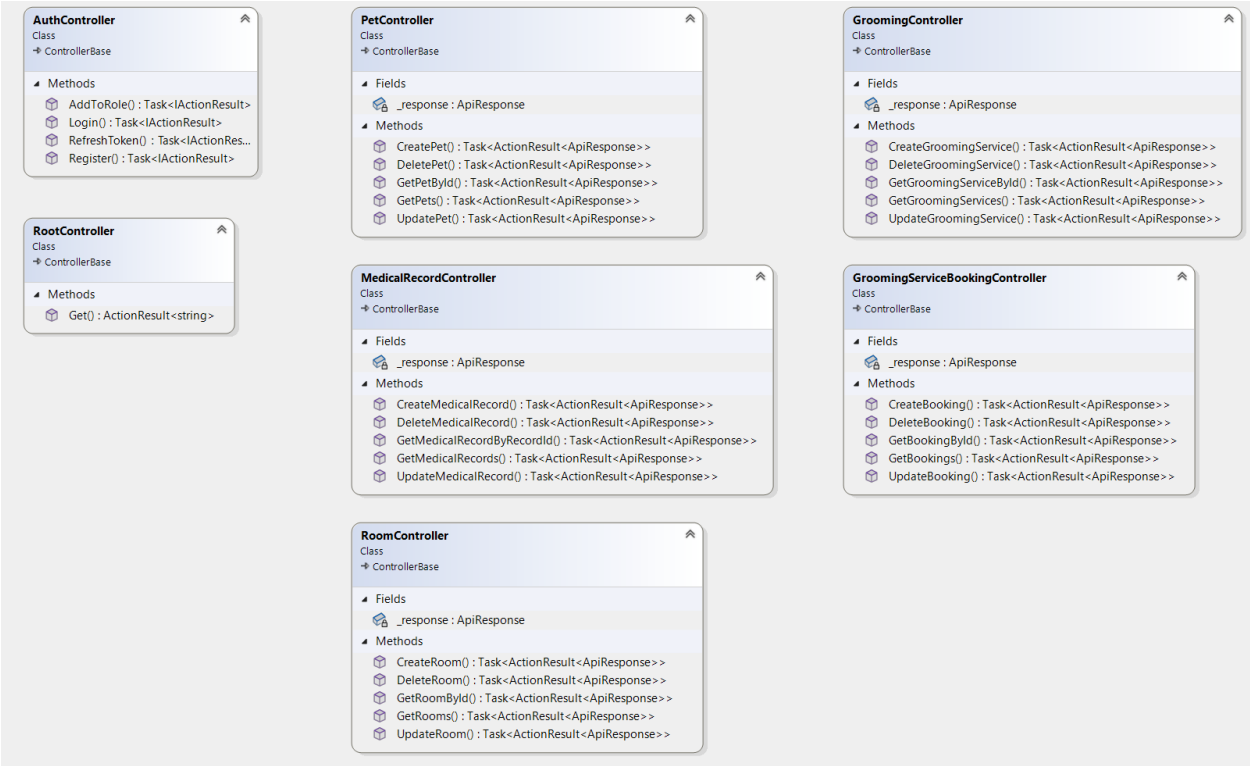
- Repository sử dụng Data Context để truy cập các DbSet tương ứng với từng loại thực thể.

- Repository thực hiện các thao tác như thêm, sửa, xóa, và truy xuất dữ liệu thông qua Data Context.

- Data Context quản lý vòng đời của các kết nối và đảm bảo tính nhất quán của dữ liệu.

d) *Controllers*

Sơ đồ lớp trong package Controllers:



Hình 3.3.5-4 Sơ đồ lớp trong package Controllers

3.3.5.d.1 Giao tiếp với các thành phần của các package khác

Giao tiếp với Services

Vai trò: Controllers sử dụng các service để thực hiện logic nghiệp vụ. Các service này được tiêm vào (injected) thông qua constructor.

Quy trình:

- Controller nhận yêu cầu HTTP từ người dùng.
- Controller gọi phương thức tương ứng của service để xử lý yêu cầu.
- Service thực hiện logic nghiệp vụ, có thể truy xuất hoặc cập nhật dữ liệu thông qua repository.
- Service trả kết quả về cho controller.
- Controller trả kết quả về cho người dùng dưới dạng HTTP response.

Giao tiếp với Models và DTOs

Vai trò: Controllers nhận DTOs từ người dùng thông qua yêu cầu HTTP và chuyển các DTOs này vào service. Các service sẽ chuyển đổi DTOs thành models nếu cần và ngược lại trước khi trả kết quả về cho controller.

Quy trình:

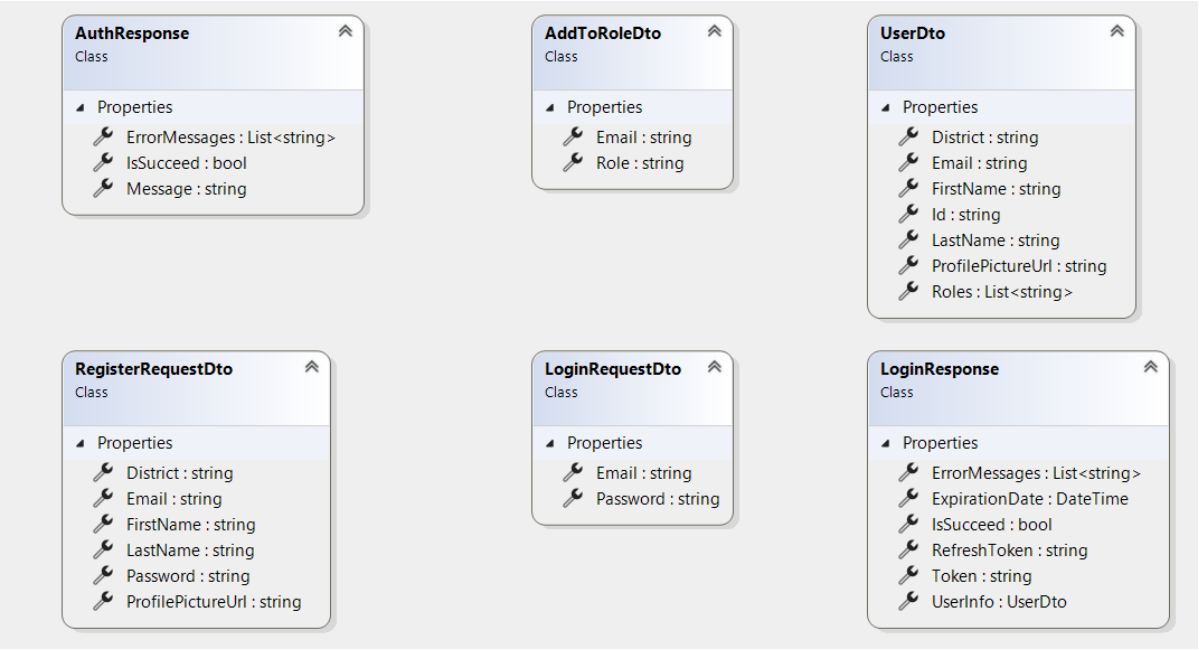
- Controller nhận DTO từ yêu cầu HTTP.
- Controller chuyển DTO vào service để xử lý.
- Service chuyển đổi DTO thành model (nếu cần), xử lý dữ liệu và có thể trả về DTO khác.
- Controller trả DTO kết quả về cho người dùng dưới dạng HTTP response.
- Giao tiếp với Repositories
- Controllers không trực tiếp giao tiếp với repositories; thay vào đó, chúng sử dụng services để truy xuất dữ liệu từ repositories. Điều này đảm bảo rằng logic truy xuất dữ liệu được tách biệt khỏi logic xử lý yêu cầu HTTP.

e) *DTOs*

DTOs chứa các class đại diện cho dữ liệu được gửi đến từ người dùng đến hệ thống và dữ liệu được trả về sau khi xử lí từ hệ thống.

Mỗi Models sẽ có một số DTO tương ứng để thực hiện các thao tác xem, thêm, sửa.

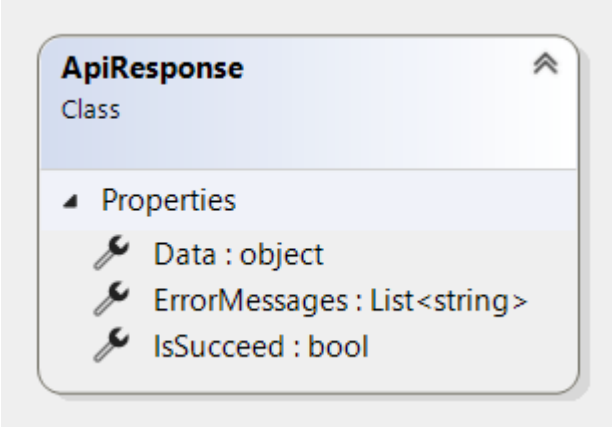
Ví dụ, các DTO tương ứng với các request về xác thực người dùng như sau:



Hình 3.3.5-5 DTOs

LoginRequestDto là cấu trúc mà một request gửi đến địa chỉ /api/auth/login để nhận về token xác thực người dùng từ hệ thống. Request bao gồm thông tin đăng nhập.

DTO cho các response trả về từ API như dưới đây:



Hình 3.3.5-6 LoginRequestDto

Khi người dùng (client) gửi một request đến API để đọc dữ liệu hoặc thêm, sửa, xoá dữ liệu thì sẽ nhận về một response định dạng JSON có cấu trúc tương tự như lớp ApiResponse. Trong đó giá trị IsSucceed thể hiện rằng thao tác có thành công hay không, nếu IsSucceed = false thì các lỗi sẽ được trả về ở ErrorMessage, ngược lại nếu thành công thì dữ liệu được yêu cầu (nếu có) sẽ được gửi ở phần Data.

3.3.6 Xác thực và xác quyền người dùng trong hệ thống

Hệ thống quản lý trung tâm chăm sóc thú cưng sử dụng JSON Web Tokens (JWT) để xác thực và xác quyền người dùng. JWT là một tiêu chuẩn mở (RFC 7519) định dạng token để truyền tải thông tin giữa các bên một cách an toàn dưới dạng JSON object. Hệ thống cũng sử dụng refresh token cùng với access token để tăng cường bảo mật và cải thiện trải nghiệm người dùng.

a) ***Xác thực người dùng***

3.3.6.a.1 Đăng nhập và cấp phát token

Quy trình đăng nhập:

- Người dùng gửi yêu cầu đăng nhập với thông tin xác thực (email và password) đến API endpoint.

- Server xác minh thông tin xác thực của người dùng.

- Nếu thông tin chính xác, server sẽ tạo và gửi lại cho người dùng một access token và một refresh token.

Access Token:

- Thời gian hiệu lực ngắn: Access token có thời gian hiệu lực ngắn (thường từ 5 đến 30 phút) để giảm thiểu rủi ro khi bị lộ.

- Payload: Chứa các thông tin xác thực như userId, roles, và các claims khác cần thiết cho việc xác thực và xác quyền.

Refresh Token:

- Thời gian hiệu lực dài hơn: Refresh token có thời gian hiệu lực dài hơn (thường từ vài giờ đến vài ngày) để giảm tần suất người dùng phải đăng nhập lại.

- Sử dụng để lấy Access Token mới: Khi access token hết hạn, người dùng có thể sử dụng refresh token để yêu cầu server trả về một access token mới mà không cần phải đăng nhập lại.

b) ***Quy trình xác thực người dùng***

- Một khoá bí mật được lưu ở server, mỗi khi một access token mới được tạo ra để trả về cho client muốn đăng nhập thì server sẽ thực hiện ký vào token đó sử dụng khoá bí mật đó.

- Khi client muốn truy cập vào một tài nguyên yêu cầu xác thực (hầu hết tài nguyên trong hệ thống đều yêu cầu xác thực) thì client cần thêm access token được trả về từ server sau khi đăng nhập thành công vào phần header của HTTP request.

- Khi server nhận được request đó, đầu tiên nó xác thực tính toàn vẹn và tính xác thực của access token bằng cách kiểm tra nó sử dụng cùng một khoá bí mật được lưu ở server (mã khoá đối xứng).

- Nếu quá trình xác thực thành công và server xác nhận access token vẫn còn thời gian hiệu lực thì client sẽ được xác thực và server trả về tài nguyên mà client yêu cầu (nếu thành công).

- Nếu quá trình xác thực thất bại vì lí do access token không toàn vẹn hoặc hết thời gian hiệu lực thì server sẽ gửi một response với mã lỗi 401 Unauthorized.

c) *Quy trình làm mới Access Token*

Yêu cầu làm mới token:

- Khi access token hết hạn, người dùng gửi yêu cầu làm mới token kèm với refresh token đến API endpoint. Điều này giúp client không cần phải yêu cầu người dùng nhập lại thông tin đăng nhập (email và mật khẩu) để gửi về /login và yêu cầu server tạo một access token mới.

- Server xác minh refresh token. Nếu hợp lệ, server sẽ tạo và gửi lại cho người dùng một access token mới.

Lưu trữ và quản lý refresh token:

- Refresh token được lưu trữ an toàn trên server.

- Refresh token sẽ bị vô hiệu hoá một khi nó được sử dụng.

- Refresh token có thể bị thu hồi nếu người dùng đăng xuất hoặc nếu có dấu hiệu bị lộ.

d) ***Xác quyền người dùng***

Phân quyền dựa trên vai trò:

- Mỗi người dùng có thể có một hoặc nhiều vai trò (roles) như Admin, User.

- Vai trò của người dùng được mã hóa trong access token dưới dạng các claims.

Kiểm tra quyền truy cập:

- Khi người dùng gửi yêu cầu tới một API endpoint, hệ thống sẽ kiểm tra access token để xác định danh tính và vai trò của người dùng.

- Dựa trên các claims trong token, hệ thống sẽ xác định người dùng có quyền thực hiện hành động yêu cầu hay không.

- Nếu người dùng không có quyền để thực hiện tác vụ thì hệ thống gửi về một response với mã lỗi 403 Forbidden.

Middleware xác thực và xác quyền:

- Các middleware cung cấp bởi ASP.NET được sử dụng để tự động kiểm tra và xác minh access token cho mỗi yêu cầu.

- Middleware cũng kiểm tra các claims để xác định quyền truy cập của người dùng đối với các tài nguyên và hành động cụ thể.

3.4 Triển khai hệ thống

Trong dự án quản lý trung tâm chăm sóc thú cưng, hệ thống được triển khai với các thành phần chính bao gồm back-end API, cơ sở dữ liệu, và front-end web. Mỗi thành phần được host trên các nền tảng khác nhau nhằm tối ưu hóa hiệu suất và khả năng mở rộng của hệ thống.

a) ***Cấu trúc triển khai***

3.4.1.a.1 Back-end API

Nền tảng: Azure Web App

Công nghệ: .NET (hoặc ngôn ngữ và framework khác được sử dụng cho API)

Chức năng: Cung cấp các API endpoint để xử lý logic nghiệp vụ, quản lý xác thực và xác quyền người dùng, và giao tiếp với cơ sở dữ liệu.

3.4.1.a.2 Cơ sở dữ liệu

Nền tảng: Azure SQL Server

Công nghệ: SQL Server

Chức năng: Lưu trữ dữ liệu về người dùng, thú cưng, lịch khám và chăm sóc, hồ sơ y tế, dịch vụ chăm sóc, và các thông tin khác của hệ thống.

3.4.1.a.3 Front-end Web

Nền tảng: Vercel

Công nghệ: React

Chức năng: Giao diện người dùng, cung cấp các trang web để người dùng tương tác với hệ thống, gửi yêu cầu đến API và hiển thị dữ liệu từ API.

b) ***Quy trình triển khai (Deployment Process)***

3.4.1.b.1 Back-end API trên Azure Web App

Thiết lập Azure Web App:

- Tạo một App Service trên Azure.

- Cấu hình App Service với các thiết lập cần thiết như SKU, scaling, và môi trường runtime (.NET 8.0).

Triển khai mã nguồn:

- Sử dụng GitHub Actions là một công cụ CI/CD để tự động hóa quy trình build và deploy mã nguồn API lên Azure Web App.

- Cấu hình pipeline để thực hiện các bước như build, run unit tests, và deploy lên App Service.

Cấu hình môi trường:

- Thiết lập các biến môi trường (environment variables) cần thiết cho ứng dụng, như kết nối cơ sở dữ liệu, các khóa API, và các thiết lập cấu hình khác.

- Cấu hình SSL/TLS để đảm bảo bảo mật trong giao tiếp giữa client và server.

3.4.1.b.2 Cơ sở dữ liệu trên Azure SQL Server

Thiết lập Azure SQL Server:

- Tạo một Azure SQL Server instance.

- Tạo một cơ sở dữ liệu mới trên SQL Server instance.

Triển khai cơ sở dữ liệu:

- Sử dụng công cụ Azure Data Studio để quản lý cơ sở dữ liệu.

- Cấu hình firewall rules để cho phép các ứng dụng và dịch vụ khác truy cập cơ sở dữ liệu.

Bảo mật và sao lưu:

- Thiết lập các chính sách bảo mật như Transparent Data Encryption (TDE) và Azure SQL Threat Detection.

- Thiết lập sao lưu tự động và kế hoạch phục hồi để đảm bảo dữ liệu được bảo vệ và có thể phục hồi trong trường hợp xảy ra sự cố.

3.4.1.b.3 Front-end Web trên Vercel

Thiết lập Vercel project:

- Tạo một project mới trên Vercel và kết nối với repository của mã nguồn front-end trên GitHub, GitLab, hoặc Bitbucket.

Triển khai mã nguồn:

- Sử dụng các thiết lập tự động của Vercel để build và deploy ứng dụng React mỗi khi có thay đổi được đẩy lên repository.

- Cấu hình các biến môi trường cần thiết cho ứng dụng React.

Quản lý tên miền:

- Cấu hình tên miền tùy chỉnh (nếu có) trên Vercel để trỏ đến ứng dụng React.

- Cấu hình SSL/TLS để đảm bảo bảo mật cho website.

- Kết nối các thành phần.

3.4.1.b.4 Kết nối Back-end API và Cơ sở dữ liệu:

- Back-end API trên Azure Web App sẽ sử dụng các biến môi trường để lưu trữ chuỗi kết nối (connection string) tới Azure SQL Server.

- Giao tiếp giữa API và SQL Server sẽ được bảo mật bằng cách sử dụng SSL/TLS.

3.4.1.b.5 Kết nối Front-end Web và Back-end API:

- Front-end React trên Vercel sẽ gửi các yêu cầu HTTP tới các endpoint của Back-end API.

- Các URL của API sẽ được cấu hình trong các biến môi trường của ứng dụng React để dễ dàng thay đổi khi cần thiết.

3.4.1.b.6 Bảo mật

- Sử dụng HTTPS để mã hóa giao tiếp giữa client và server.

- Cấu hình các biện pháp bảo mật như OAuth, JWT cho xác thực và xác quyền người dùng.

3.5 Cài đặt và chạy ứng dụng

a) *Công cụ cần thiết*

Để triển khai, cần cài đặt các phần mềm sau để chạy:

- Visual Studio 2022 (hoặc mới hơn), trong quá trình cài đặt, chọn workload: ASP.NET and web development

- .NET 8 SDK (Dùng để build và chạy các ứng dụng ASP.NET Core)

- Node.js v16 trở lên (Cần thiết cho các dự án có sử dụng frontend framework như React hoặc Angular)

- SQL Server (Dùng để lưu trữ và quản lý dữ liệu)

- SQL Server Management Studio (SSMS) (Dùng để thao tác và quản lý cơ sở dữ liệu SQL Server)

- Git (Dùng để quản lý mã nguồn và làm việc với GitHub hoặc GitLab)

b) ***Các bước cài đặt và chạy project***

Bước 1: Tải mã nguồn

- Mở Visual Studio → File → Open → chọn file .sln của project

(Git: git clone https://github.com/tlambvq2/ASPNET-DK23TTC11-tranlam-petcare_at_home.git)

Bước 2: Khởi động backend

- Mở thử mục PetCareSystem.

cd PetCareSystem

- Mở file PetShopProj.sln bằng Visual Studio

- Ấn F5 hoặc Ctrl + F5 để khởi động.

Bước 3: Khởi động frontend

- Mở thử mục PCS_front_end.

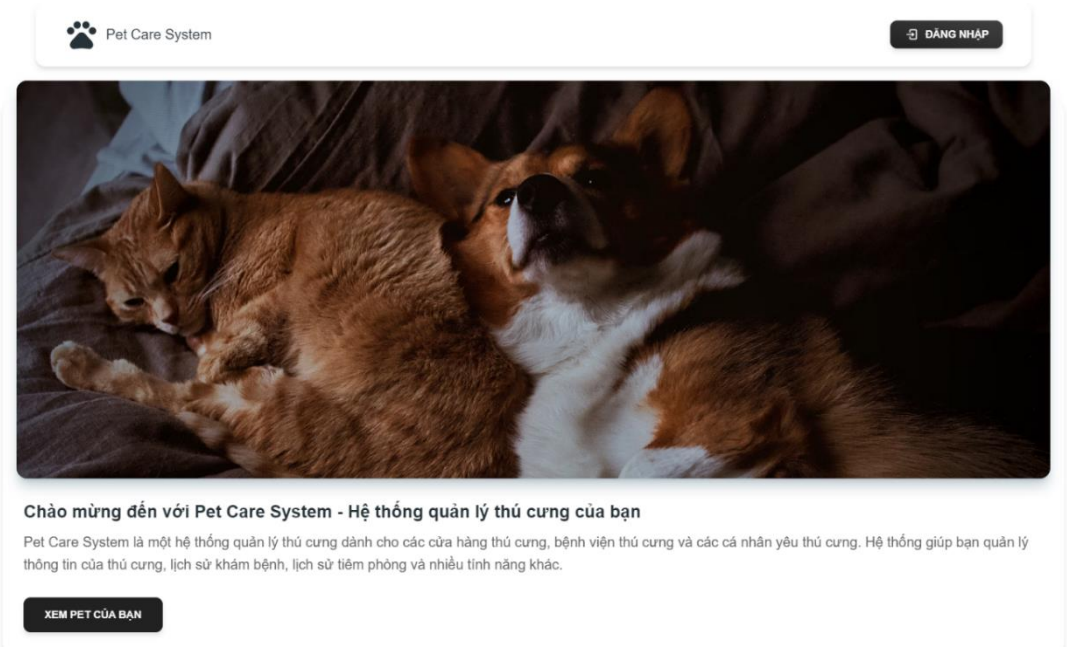
cd PCS_front_end

npm install

npm run dev

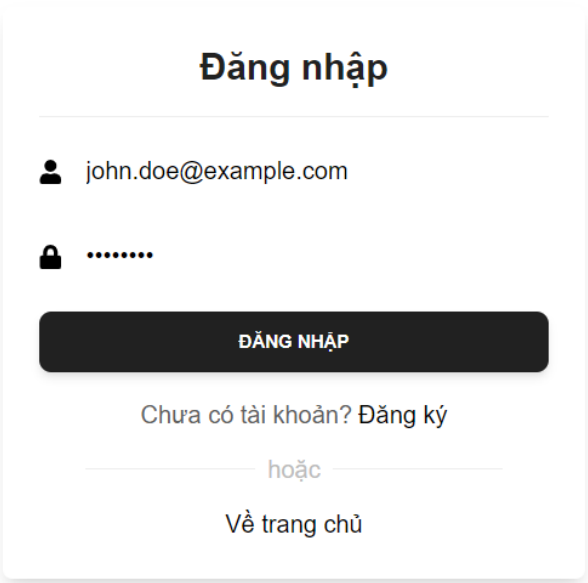
CHƯƠNG 4 KẾT LUẬN

4.1 Giao diện trang chủ



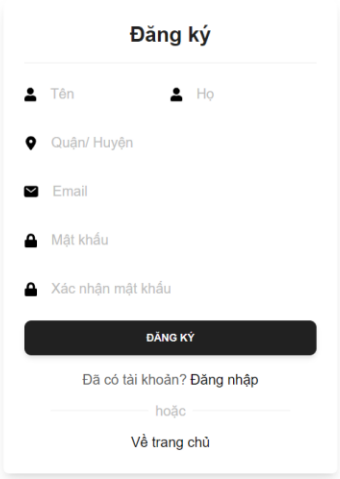
Hình 3.3.6-1 Trang chủ

4.2 Giao diện trang login



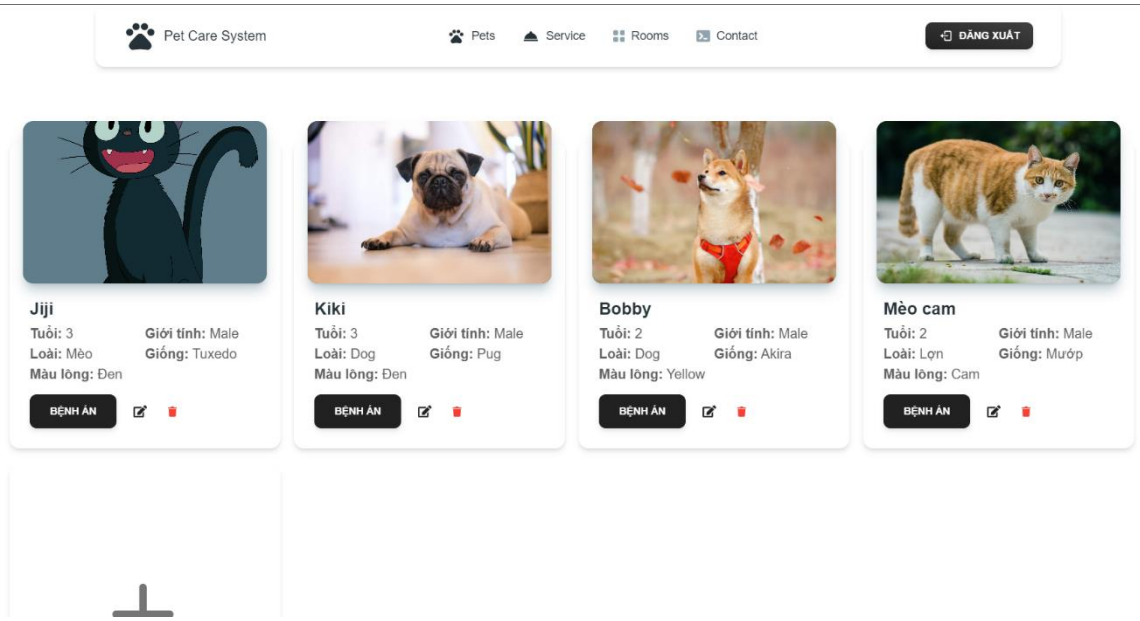
Hình 3.3.6-1 Login

4.3 Giao diện trang đăng ký



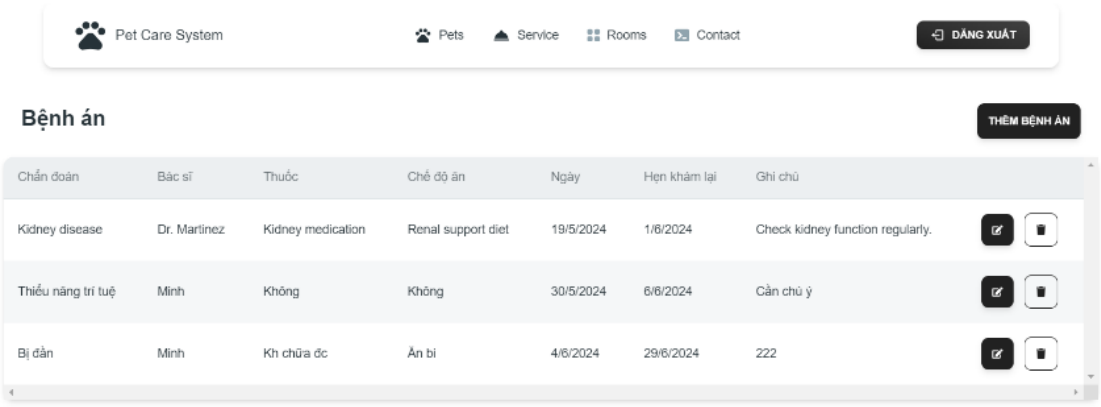
Hình 3.3.6-1 Đăng ký

4.4 Danh sách các Pet



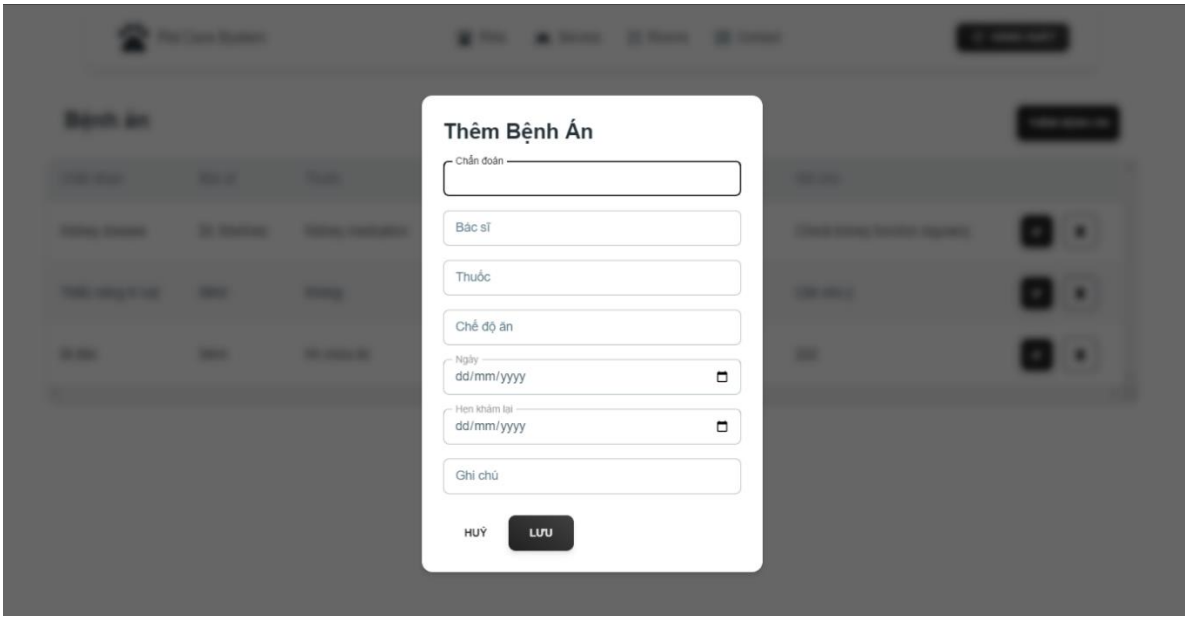
Hình 3.3.6-1 Danh sách các pet

- Xem bệnh án của Pet:



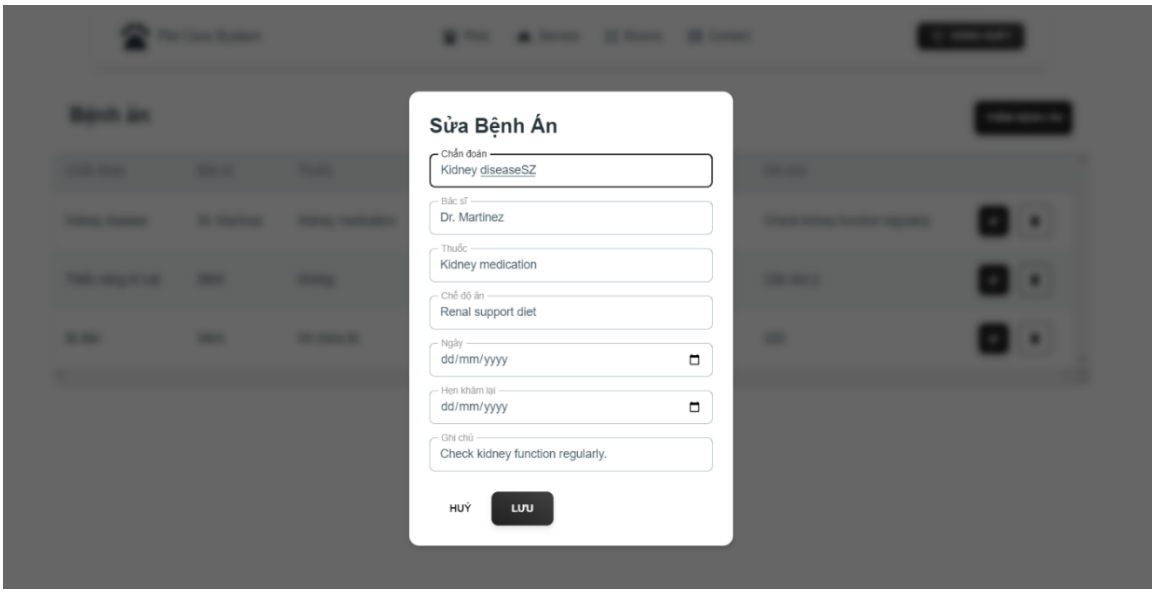
Hình 3.3.6-2 Xem bệnh án của Pet

- Thêm bệnh án:



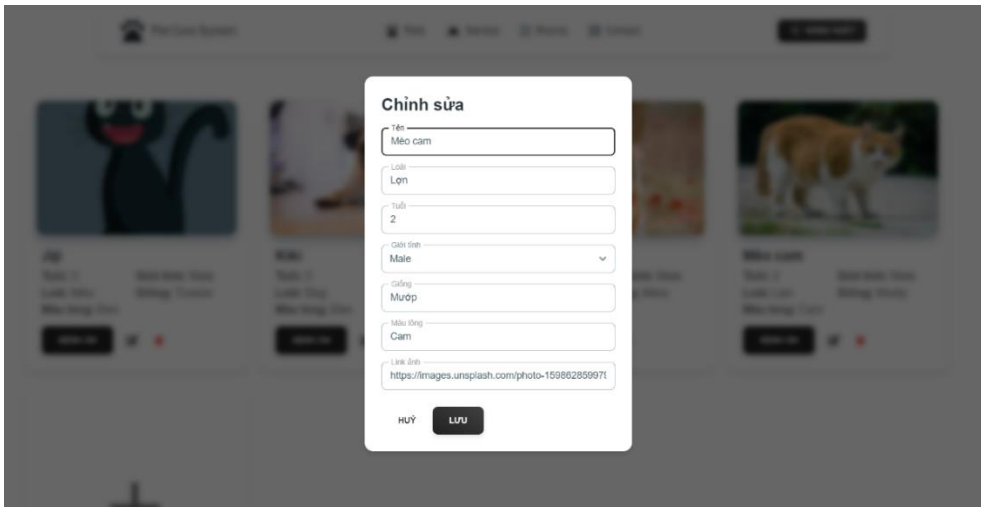
Hình 3.3.6-3 Thêm bệnh án

- Sửa bệnh án:



Hình 3.3.6-4 Sửa bệnh án

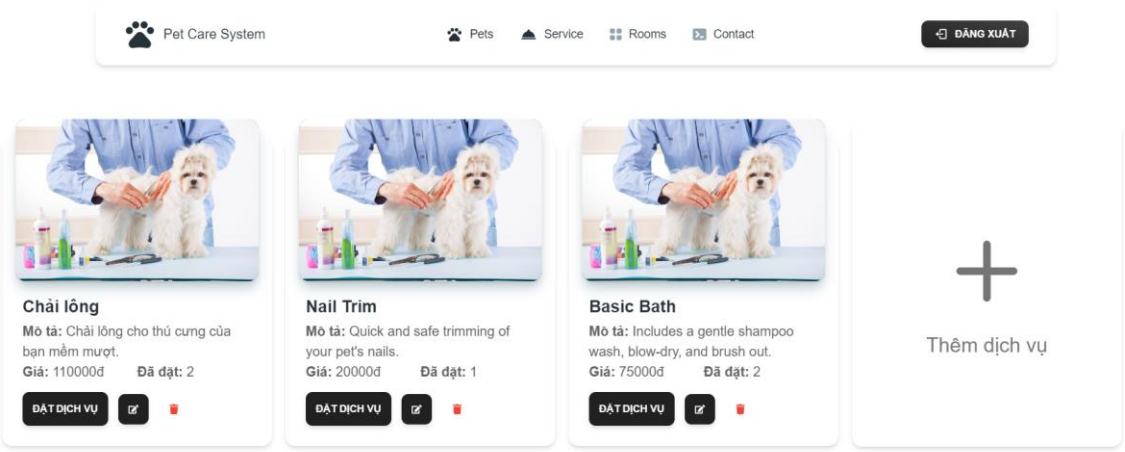
- Thay đổi thông tin của Pet:



Hình 3.3.6-5 Thay đổi thông tin Pet

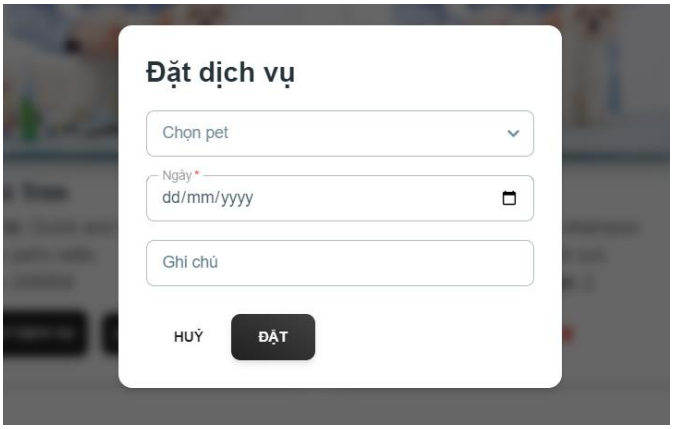
4.5 Trang quản lý các dịch vụ và phòng thuê

Danh sách các dịch vụ:



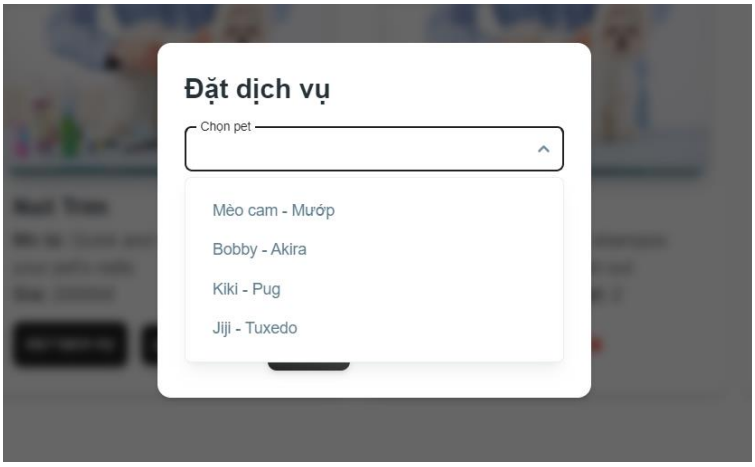
Hình 3.3.6-1 Danh sách các dịch vụ

- Đặt dịch vụ:



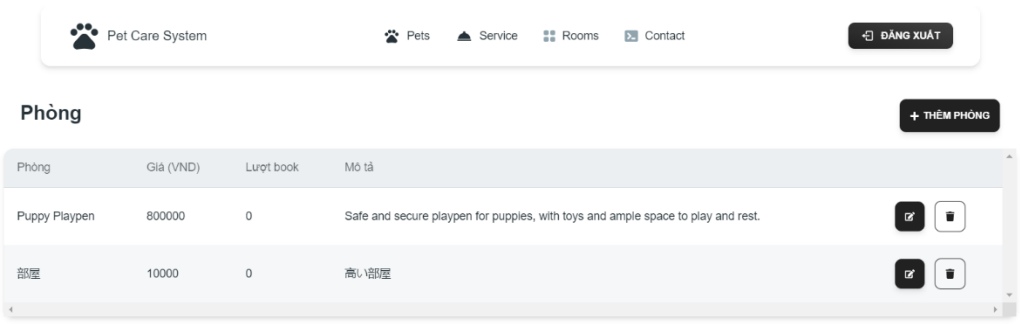
Hình 3.3.6-2 Đặt dịch vụ

- Chọn Pet:



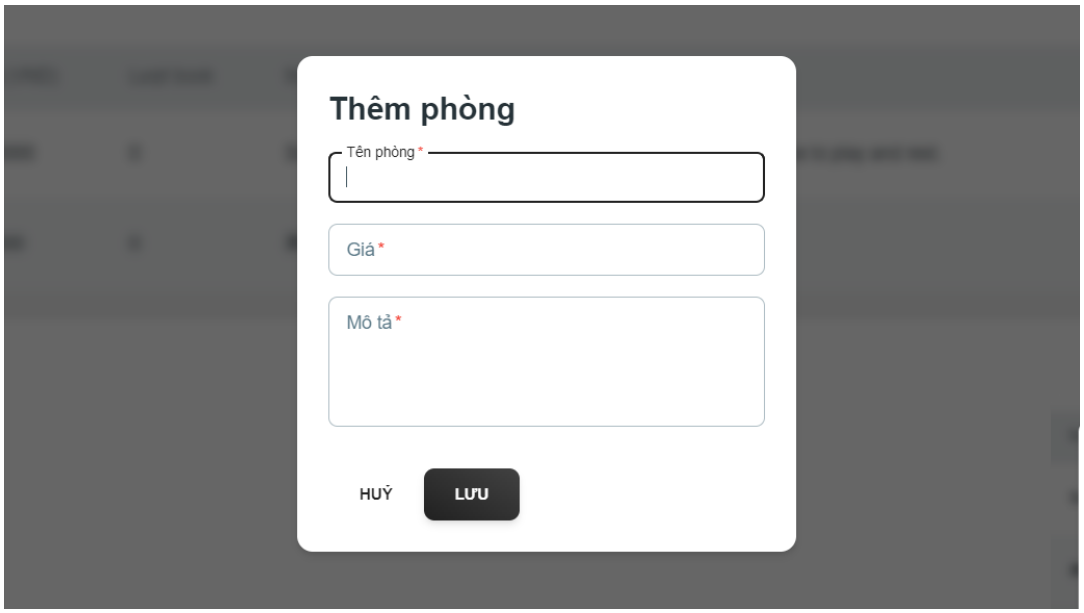
Hình 3.3.6-3 Chọn Pet

Trang quản lí các phòng:



Hình 3.3.6-4 Trang quản lý các phòng

- Thêm phòng:



Hình 3.3.6-5 Thêm phòng

CHƯƠNG 5 HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Hệ thống quản lý trung tâm chăm sóc thú cưng được xây dựng đã đáp ứng được các yêu cầu chức năng cơ bản như quản lý thú cưng, hồ sơ y tế, đặt dịch vụ và phân quyền người dùng. Việc kết hợp giữa ASP.NET Core, React và SQL Server mang lại một giải pháp hiệu quả, dễ mở rộng và có thể triển khai thực tế trên nền tảng đám mây. Mặc dù hệ thống vẫn còn một số hạn chế cần cải thiện, kết quả đạt được đã phần nào chứng minh tính khả thi và ứng dụng thực tiễn của mô hình phát triển hiện đại.

5.2 Hạn chế

- Giao diện người dùng chưa thực sự tối ưu cho trải nghiệm trên thiết bị Chưa tích hợp chức năng thanh toán trực tuyến cho các dịch vụ.
- Hệ thống thống kê còn đơn giản, chưa hỗ trợ biểu đồ phân tích nâng cao.
- Tính năng gửi thông báo (email/sms) chưa được tích hợp.
- Chưa có hệ thống log chi tiết để theo dõi hoạt động và lỗi hệ thống.

5.3 Hướng phát triển

- Phát triển giao diện tương thích hoàn toàn với thiết bị di động (responsive + mobile-first).
- Tích hợp cổng thanh toán online như VNPay, Momo hoặc Stripe.

Bổ sung hệ thống biểu đồ báo cáo nâng cao sử dụng thư viện charting (như Chart.js hoặc Recharts).

- Triển khai tính năng gửi thông báo lịch hẹn qua email hoặc tin nhắn SMS.
- Phát triển dashboard quản trị nâng cao với thống kê theo thời gian thực.
- Cải thiện bảo mật nâng cao: xác thực 2 lớp (2FA), log hoạt động người dùng.
- Mở rộng hệ thống API để dễ tích hợp với ứng dụng mobile hoặc nền tảng khác.
- Xây dựng ứng dụng mobile tương thích (React Native hoặc Flutter) để hỗ trợ người dùng tiện lợi hơn.

DANH MỤC TÀI LIỆU THAM KHẢO

Tài liệu tham khảo:

[1] Microsoft Docs – ASP.NET Core Overview,

https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0

[2] Microsoft Docs – ASP.NET Core Performance Best Practices,

https://learn.microsoft.com/en-us/aspnet/core/performance/?view=aspnetcore-8.0

[3] Microsoft Docs – Publish and host ASP.NET Core apps,

https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-8.0

[4] Microsoft Azure – Host ASP.NET Core App in Azure App Service,

https://learn.microsoft.com/en-us/azure/app-service/quickstart-dotnetcore?tabs=net80&pivots=development-environment-vs

PHỤ LỤC