# Vet-Health — Backend (.NET 9)

Documentação de arquitetura, setup e convenções do back-end do projeto **Vet-Health**.

**Resumo:** Foi estruturada uma solução em camadas com .NET 9, Entity Framework Core (SQL Server), ASP.NET Identity e Swagger. Foram definidas entidades de domínio, DbContext, configuração de DI, connection strings e fluxo de migrations.

## Sumário

* [Visão Geral](#visão-geral)
* [Estrutura da Solução](#estrutura-da-solução)
* [Racional das Camadas](#racional-das-camadas)
* [Versões e Requisitos](#versões-e-requisitos)
* [Dependências NuGet](#dependências-nuget)
* [Convenções de Nomes e Namespaces](#convenções-de-nomes-e-namespaces)
* [Domínio (Entities)](#domínio-entities)
* [Infraestrutura (Identity + EF Core)](#infraestrutura-identity--ef-core)
  + [ApplicationUser](#applicationuser)
  + [VetHealthDbContext](#vethealthdbcontext)
* [API (Program.cs e Pipeline)](#api-programcs-e-pipeline)
* [Connection Strings](#connection-strings)
* [Migrations e Banco de Dados](#migrations-e-banco-de-dados)
* [Swagger (Documentação Interativa)](#swagger-documentação-interativa)
* [Erros Comuns e Soluções](#erros-comuns-e-soluções)
* [Próximos Passos](#próximos-passos)

## Visão Geral

Foi adotado um design em camadas, isolando regras de negócio do acesso a dados e da camada HTTP. O banco alvo é **SQL Server** e a autenticação foi preparada com **ASP.NET Identity** usando **GUID** como chave primária. O EF Core foi configurado para gerar migrations no projeto **Infrastructure** enquanto usa a configuração (DI e connection string) do projeto **Api** como *startup*.

## Estrutura da Solução

vet-health/  
├─ vet-health.Api/ # Camada de apresentação: Program.cs, Controllers, Swagger  
├─ vet-health.Application/ # Contratos, DTOs, casos de uso (sem dependência de infra)  
├─ vet-health.Domain/ # Entidades e regras de negócio puras  
└─ vet-health.Infrastructure/ # EF Core, Identity, repositórios, mapeamentos

Foi estabelecida a referência entre projetos da seguinte forma:

* **Api → Infrastructure**
* **Infrastructure → Domain**
* **Application → Domain**
* **Domain** sem dependências para manter o núcleo do negócio independente.

## Racional das Camadas

* **Domain:** concentração das entidades e regras de negócio; não foi incluída dependência de EF Core aqui para manter independência e testabilidade.
* **Application:** definição de casos de uso, interfaces e DTOs; atua como fronteira entre HTTP e domínio.
* **Infrastructure:** implementação técnica (EF Core, Identity, repositórios, providers externos).
* **Api:** camada HTTP mínima, registra serviços (DI), expõe endpoints e integra Swagger.

Este arranjo promove baixo acoplamento, testes mais simples e possibilidade de troca de tecnologias (por exemplo, alterar o provedor de banco) com impacto localizado.

## Versões e Requisitos

* **.NET SDK:** 9.0+
* **Banco:** SQL Server (local/contêiner/instância remota)
* **Ferramenta:** dotnet-ef instalada globalmente para migrations

dotnet tool update -g dotnet-ef

## Dependências NuGet

### vet-health.Api

* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Design ← necessário para ferramentas do EF no *startup project*
* Swashbuckle.AspNetCore ← habilita Swagger

### vet-health.Infrastructure

* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Design ← necessário para geração de migrations
* Microsoft.AspNetCore.Identity.EntityFrameworkCore ← Identity + EF Core

Observação: a presença de Microsoft.EntityFrameworkCore.Design **na Api e na Infrastructure** foi necessária para suportar o fluxo de migrations com -s (startup) e -p (project do DbContext).

## Convenções de Nomes e Namespaces

* Os **nomes de pastas/projetos** utilizaram hífen (vet-health), porém em **namespaces** foi adotado VetHealth.\* (PascalCase, sem hífen) por clareza.
* Nos arquivos .cs, os namespaces foram padronizados, por exemplo: namespace VetHealth.Domain.Entities; e namespace VetHealth.Infrastructure.Persistence;.
* O alinhamento entre using e namespace foi tratado com atenção para evitar erros do tipo *namespace não encontrado*.

## Domínio (Entities)

Foi mantido o domínio **sem** atributos de EF quando possível. Exemplo simplificado (Pet):

namespace VetHealth.Domain.Entities;  
  
public class Pet : BaseEntity  
{  
 public Guid OwnerUserId { get; set; }  
 public string Name { get; set; } = default!;  
 public string? Species { get; set; }  
 public string? Breed { get; set; }  
 public DateTime? BirthDate { get; set; }  
 public List<Post> Posts { get; set; } = new();  
}

Benefícios: maior testabilidade, independência de infraestrutura e clareza de responsabilidade.

## Infraestrutura (Identity + EF Core)

### ApplicationUser

Foi definido um usuário de autenticação customizado para permitir extensão do perfil e uso de **GUID** como chave:

using Microsoft.AspNetCore.Identity;  
  
namespace VetHealth.Infrastructure.Identity;  
  
public class ApplicationUser : IdentityUser<Guid>  
{  
 public string? DisplayName { get; set; }  
 public string? AvatarUrl { get; set; }  
}

### VetHealthDbContext

O DbContext foi configurado herdando de IdentityDbContext<ApplicationUser, IdentityRole<Guid>, Guid> para compartilhar o mesmo banco com as tabelas do Identity:

using Microsoft.AspNetCore.Identity;  
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;  
using Microsoft.EntityFrameworkCore;  
using VetHealth.Infrastructure.Identity;  
using VetHealth.Domain.Entities;  
  
namespace VetHealth.Infrastructure.Persistence;  
  
public class VetHealthDbContext : IdentityDbContext<ApplicationUser, IdentityRole<Guid>, Guid>  
{  
 public VetHealthDbContext(DbContextOptions<VetHealthDbContext> options) : base(options) {}  
  
 public DbSet<Pet> Pets => Set<Pet>();  
 public DbSet<Post> Posts => Set<Post>();  
 public DbSet<Comment> Comments => Set<Comment>();  
 public DbSet<PostLike> PostLikes => Set<PostLike>();  
 public DbSet<VeterinarianProfile> VeterinarianProfiles => Set<VeterinarianProfile>();  
 public DbSet<Question> Questions => Set<Question>();  
 public DbSet<Answer> Answers => Set<Answer>();  
  
 protected override void OnModelCreating(ModelBuilder b)  
 {  
 base.OnModelCreating(b);  
  
 // Renomeação opcional de tabelas do Identity  
 b.Entity<ApplicationUser>().ToTable("Users");  
 b.Entity<IdentityRole<Guid>>().ToTable("Roles");  
 b.Entity<IdentityUserRole<Guid>>().ToTable("UserRoles");  
 b.Entity<IdentityUserLogin<Guid>>().ToTable("UserLogins");  
 b.Entity<IdentityUserClaim<Guid>>().ToTable("UserClaims");  
 b.Entity<IdentityUserToken<Guid>>().ToTable("UserTokens");  
 b.Entity<IdentityRoleClaim<Guid>>().ToTable("RoleClaims");  
  
 // Exemplos de constraints/índices por entidade (ajustes adicionais podem ser feitos)  
 b.Entity<Pet>(e =>  
 {  
 e.ToTable("Pets");  
 e.Property(p => p.Name).HasMaxLength(120).IsRequired();  
 e.Property(p => p.Species).HasMaxLength(60);  
 e.Property(p => p.Breed).HasMaxLength(120);  
 e.HasIndex(p => p.OwnerUserId);  
 });  
  
 b.Entity<Post>(e =>  
 {  
 e.ToTable("Posts");  
 e.Property(p => p.ContentText).HasMaxLength(2000);  
 e.Property(p => p.Hashtags).HasMaxLength(500);  
 e.HasIndex(p => p.PetId);  
 e.HasIndex(p => p.AuthorUserId);  
 });  
  
 b.Entity<Comment>(e =>  
 {  
 e.ToTable("Comments");  
 e.Property(c => c.Text).HasMaxLength(1000).IsRequired();  
 e.HasIndex(c => c.PostId);  
 e.HasIndex(c => c.AuthorUserId);  
 });  
  
 b.Entity<PostLike>(e =>  
 {  
 e.ToTable("PostLikes");  
 e.HasIndex(l => new { l.PostId, l.UserId }).IsUnique();  
 });  
  
 b.Entity<VeterinarianProfile>(e =>  
 {  
 e.ToTable("VeterinarianProfiles");  
 e.Property(v => v.CRMV).HasMaxLength(50).IsRequired();  
 e.Property(v => v.LicenseState).HasMaxLength(2);  
 e.HasIndex(v => v.UserId).IsUnique();  
 e.HasIndex(v => v.CRMV);  
 });  
  
 b.Entity<Question>(e =>  
 {  
 e.ToTable("Questions");  
 e.Property(q => q.Title).HasMaxLength(160).IsRequired();  
 e.Property(q => q.Body).HasMaxLength(4000).IsRequired();  
 e.Property(q => q.Category).HasMaxLength(60);  
 e.HasIndex(q => q.AskedByUserId);  
 e.HasIndex(q => q.Category);  
 });  
  
 b.Entity<Answer>(e =>  
 {  
 e.ToTable("Answers");  
 e.Property(a => a.Body).HasMaxLength(4000).IsRequired();  
 e.HasIndex(a => a.QuestionId);  
 e.HasIndex(a => a.AnsweredByUserId);  
 });  
 }  
}

(Opcional) Uma VetHealthDbContextFactory pode ser adicionada para facilitar o design-time quando a API ainda não estiver configurada.

## API (Program.cs e Pipeline)

Foi registrada a configuração de DI para DbContext, Identity e Swagger. Exemplo mínimo sugerido:

using Microsoft.AspNetCore.Identity;  
using Microsoft.EntityFrameworkCore;  
using VetHealth.Infrastructure.Identity;  
using VetHealth.Infrastructure.Persistence;  
  
var builder = WebApplication.CreateBuilder(args);  
  
var conn = builder.Configuration.GetConnectionString("DefaultConnection");  
  
builder.Services.AddDbContext<VetHealthDbContext>(opt =>  
 opt.UseSqlServer(conn));  
  
builder.Services  
 .AddIdentity<ApplicationUser, IdentityRole<Guid>>(opt =>  
 {  
 opt.Password.RequiredLength = 6;  
 opt.Password.RequireNonAlphanumeric = false;  
 opt.Password.RequireUppercase = false;  
 opt.Password.RequireLowercase = false;  
 opt.Password.RequireDigit = false;  
 opt.User.RequireUniqueEmail = true;  
 })  
 .AddEntityFrameworkStores<VetHealthDbContext>()  
 .AddDefaultTokenProviders();  
  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
  
var app = builder.Build();  
  
if (app.Environment.IsDevelopment())  
{  
 app.UseSwagger();  
 app.UseSwaggerUI();  
}  
  
app.UseHttpsRedirection();  
app.UseAuthentication();  
app.UseAuthorization();  
app.MapControllers();  
app.Run();

## Connection Strings

A connection string foi colocada no appsettings.Development.json da API para ambiente local. Exemplo:

{  
 "ConnectionStrings": {  
 "DefaultConnection": "Server=localhost;Database=VetHealthDb;Trusted\_Connection=True;TrustServerCertificate=True;"  
 }  
}

Para ambientes de produção, recomenda-se o uso de **User Secrets** (dev) e **variáveis de ambiente/KeyVault** (prod), evitando versionamento de segredos.

## Migrations e Banco de Dados

Foi definido que as migrations residem no projeto **Infrastructure**, consumindo a configuração do projeto **Api**. Por isso, foram usados os parâmetros -p (onde está o DbContext) e -s (projeto de startup) nos comandos.

### CLI

dotnet ef migrations add InitialCreate -s "vet-health.Api" -p "vet-health.Infrastructure"  
dotnet ef database update -s "vet-health.Api" -p "vet-health.Infrastructure"

### Package Manager Console (Visual Studio)

Add-Migration InitialCreate -StartupProject "vet-health.Api" -Project "vet-health.Infrastructure"  
Update-Database -StartupProject "vet-health.Api" -Project "vet-health.Infrastructure"

### Verificações úteis

dotnet ef dbcontext list -s "vet-health.Api" -p "vet-health.Infrastructure"

## Swagger (Documentação Interativa)

Swagger foi habilitado em ambiente de desenvolvimento para facilitar testes e inspeção de endpoints. Em produção, a exposição deve ser avaliada considerando segurança e políticas internas.

## Erros Comuns e Soluções

**1)** `\*\*/\*\*\`\*\* ausentes\*\*

* *Sintoma:* métodos não reconhecidos no Program.cs.
* *Causa:* pacote Swashbuckle.AspNetCore ausente na **Api**.
* *Ação:* instalar o pacote na Api.

**2)** ``\*\* não reconhecido\*\*

* *Sintoma:* using Microsoft.EntityFrameworkCore não compila na **Api**.
* *Causa:* pacotes EF Core ausentes na **Api**.
* *Ação:* instalar Microsoft.EntityFrameworkCore e Microsoft.EntityFrameworkCore.SqlServer na Api.

**3) ``**\*\* exigido pelo EF Tools\*\*

* *Sintoma:* erro informando que o startup não referencia o pacote de Design.
* *Causa:* pacote ausente na **Api** (ou na Infrastructure).
* *Ação:* instalar Microsoft.EntityFrameworkCore.Design na Api **e** na Infrastructure.

**4) Namespaces inconsistentes**

* *Sintoma:* namespace X não existe no Y.
* *Causa:* using divergente do namespace declarado no arquivo.
* *Ação:* padronizar namespaces como VetHealth.\* e ajustar todos os using.

\*\*5) \*\*``

* *Sintoma:* EF não encontra o contexto.
* *Causa:* parâmetro -p apontando para projeto errado, projeto não compila, ou DbContext não público.
* *Ação:* compilar a solução, checar -p correto e visibilidade da classe.

**6) Migrations geradas no projeto errado**

* *Sintoma:* pasta Migrations criada na Api.
* *Causa:* comando executado sem -p apontando para a Infrastructure.
* *Ação:* remover pasta incorreta e recriar migrations com -p "vet-health.Infrastructure".

## Próximos Passos

* Implementação de **Autenticação JWT** (registro/login), roles (usuario, veterinario, admin) e *refresh tokens*.
* *Seed* inicial de roles e, se necessário, de usuários administrativos.
* Criação de **DTOs** e **Controllers** para módulos iniciais (Pets, Posts, Q&A), com validação.
* Definição de **padrões de branch/PR**, convenções de migrations (YYYYMMDDHHmm\_Descricao) e cobertura de testes.
* Externalização de segredos com **User Secrets** (dev) e **variáveis de ambiente/KeyVault** (prod).