Título: Desenvolvimento de uma API para Gerenciamento de Clínica Veterinária

Autor: Miguel Elias Bernardes Instituição: Ulbra Curso: ADS

Data: 28/05/2025

Introdução

Este artigo descreve o desenvolvimento de uma API para gerenciamento de uma clínica veterinária. O objetivo é apresentar como os requisitos funcionais e não funcionais foram implementados utilizando a tecnologia .NET 9.0, seguindo as boas práticas de desenvolvimento de software.

Desenvolvimento

Requisitos Funcionais

Cadastro de Pets: Implementado através do controlador PetsController, que permite operações CRUD (Create, Read, Update, Delete) para os registros de animais.

Exemplo de Código:

```
[HttpPost]      You, 3 hours ago • ft ...
public async Task<IActionResult> Create(Pet pet)
{
    if (!ModelState.IsValid) return BadRequest(ModelState);
    var novoPet = await _repository.Create(pet);
    return CreatedAtAction(nameof(GetById), new { id = novoPet.Id }, novoPet);
}
```

Cadastro de Clientes: O TutoresController gerencia as informações dos clientes, permitindo a inserção, edição, exclusão e listagem dos mesmos.

Exemplo de Código:

```
[HttpPut("{id}")]      You, 3 hours ago • ft
public async Task<IActionResult> Update(int id, Tutor tutor)
{
    if (id != tutor.Id) return BadRequest();
    await _repository.Update(tutor);
    return NoContent();
}

[HttpDelete("{id}")]
public async Task<IActionResult> Delete(int id)
{
    var deleted = await _repository.Delete(id);
    return deleted ? NoContent() : NotFound();
}
```

Requisitos Não Funcionais

Persistência de Dados: Utilização do Entity Framework Core para mapeamento objeto-relacional, facilitando a interação com o banco de dados SQLite (veterinaria.db).

Exemplo de Código:

```
using Microsoft.EntityFrameworkCore;
using VeterinariaAPI.Models;

namespace VeterinariaAPI.Data
{
    public class AppDbContext : DbContext      You, 3 hours ago • ft ...
    {
        public AppDbContext(DbContextOptions options) : base(options) { }

        public DbSet<Tutor> Tutores { get; set; }
        public DbSet<Pet> Pets { get; set; }
    }
}
```

Arquitetura em Camadas: A aplicação segue a arquitetura MVC (Model-View-Controller), separando responsabilidades e promovendo a manutenção do código.

Documentação da API: Implementação do Swagger para documentação automática das rotas e métodos disponíveis na API.

Validação de Dados: Utilização de Data Annotations nas classes de modelo para garantir a integridade e consistência dos dados inseridos.

Exemplo de Código:

```
using System.ComponentModel.DataAnnotations;

namespace VeterinariaAPI.Models
{
    public class Tutor
    {
        public int Id { get; set; }

        [Required]
        public string Nome { get; set; }

        [Required]
        public string Telefone { get; set; }

        public string Email { get; set; }

        public ICollection<Pet> Pets { get; set; } = new List<Pet>();
    }
}
```

Conclusão

O desenvolvimento desta aplicação proporcionou uma compreensão aprofundada sobre a construção de APIs utilizando .NET 9.0. Os principais desafios enfrentados incluíram a configuração do Entity Framework Core e a implementação de relacionamentos entre entidades. A adoção de boas práticas e padrões de projeto contribuiu para a criação de um sistema escalável e de fácil manutenção.

Referências Bibliográficas

Microsoft Docs. Documentação do ASP.NET Core.

Microsoft Docs. Documentação do Entity Framework Core.

Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.