UNIVERSIDADE LUTERANA DO BRASIL
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KAUE MARQUES MAGNUS

GERENCIADOR DE CLÍNICA VETERINÁRIA

Sistema CRUD Aplicado ao Gerenciamento de Clínicas Veterinárias

Torres
2025

KAUE MARQUES MAGNUS

GERENCIADOR DE CLÍNICA VETERINÁRIA

Sistema CRUD Aplicado ao Gerenciamento de Clínicas Veterinárias

Projeto desenvolvido afins de trabalho avaliativo AP2.

Orientador(a): Prof. Lucas Fogaça

Torres

2025

SUMÁRIO

1 INTRODUÇÃO

Este artigo tem como objetivo descrever o processo de desenvolvimento de uma aplicação web baseada em API REST voltada ao gerenciamento de uma clínica veterinária. O cenário consiste na criação de uma solução capaz de realizar o cadastro, listagem, atualização e remoção de tutores e seus respectivos pets, utilizando boas práticas de engenharia de software.

2 DESENVOLVIMENTO

Requisito Funcional 1: Cadastro de Tutores

Foi implementado um endpoint POST na controller de tutores, recebendo um objeto JSON contendo as informações do tutor. A validação básica de campos obrigatórios é realizada e o repositório é utilizado para persistência no banco SQLite.

Requisito Funcional 2: Listagem e Detalhamento de Tutores

Endpoints GET foram desenvolvidos para retornar todos os tutores ou um tutor específico por ID. Os dados são retornados em formato JSON, com seus respectivos pets associados, utilizando eager loading com Entity Framework Core.

Requisito Funcional 3: Atualização e Remoção de Tutores

Os métodos PUT e DELETE foram definidos para permitir a edição ou exclusão de tutores existentes, com verificações de existência prévias implementadas nos serviços.

Requisito Funcional 4: Gerenciamento de Pets

A estrutura implementa serviços, repositórios e controladores para pets, permitindo CRUD completo. Cada pet deve estar associado a um tutor, e a integridade relacional é mantida pelo Entity Framework.

Requisito Não Funcional 1: Padrão Repository

Toda a camada de acesso a dados foi implementada com base no padrão Repository, separando lógica de persistência do domínio da aplicação.

Requisito Não Funcional 2: Injeção de Dependência

Utilizamos injeção de dependência via construtor para serviços e repositórios, registrando os serviços em Program.cs usando o método AddScoped.

Requisito Não Funcional 3: Banco de Dados Relacional

Foi utilizado SQLite com migrations para gerar e manter o schema do banco de forma automatizada através do Entity Framework Core.

Requisito Não Funcional 4: Documentação via Swagger

A API foi documentada automaticamente com Swagger, permitindo testes e visualização interativa dos endpoints diretamente pelo navegador.

3 CONCLUSÃO

Durante o desenvolvimento da API, os principais desafios envolveram o controle de relacionamentos bidirecionais entre entidades e a separação clara de responsabilidades utilizando padrões de projeto. A aplicação das boas práticas de engenharia de software, como o uso do padrão Repository e injeção de dependência, resultou em um código mais limpo, reutilizável e de fácil manutenção.

REFERÊNCIAS

Microsoft. Entity Framework Core. https://learn.microsoft.com/ef/core/

Microsoft. ASP.NET Core Documentation. https://learn.microsoft.com/aspnet/core

Stack Overflow https://stackoverflow.com/questions