Desenvolvimento de uma API com Estrutura em Camadas Utilizando .NET 9 e Entity Framework Core

O objetivo deste artigo é apresentar a construção de uma aplicação Web API utilizando a linguagem C# com .NET 9, Entity Framework Core e banco de dados SQLite. O projeto tem como objetivo implementar uma estrutura básica de CRUD, com foco em boas práticas. O cenário abordado envolve a criação de duas entidades principais: Tutor e Pet, representando uma relação de um-para-muitos, em que um tutor pode possuir vários pets. A implementação foi realizada com o propósito de desenvolver uma estrutura de código reutilizável, testável e de fácil manutenção.

A aplicação foi desenvolvida com base em uma estrutura simples, porém eficiente, dividida em camadas: Controller, Service, Repository e Interface, todas conectadas ao AppDbContext, que representa o contexto do banco de dados. A camada Controller é responsável por receber as requisições HTTP e direcioná-las para os serviços corretos. Dentro dessa camada, são mapeadas rotas como GET, POST, PUT e DELETE para manipulação das entidades Tutor e Pet. A camada Service, por sua vez, é encarregada de conter a lógica de negócio e as validações básicas, garantindo que os dados estejam corretos antes de serem enviados à camada Repository. Esta última é responsável por executar as operações diretamente no banco de dados, utilizando o Entity Framework Core como ORM, o que reduz a complexidade das consultas SQL e torna a persistência de dados mais eficiente e segura. A Interface define os contratos das classes Service e Repository, promovendo o desacoplamento entre as camadas e permitindo maior flexibilidade.

A entidade Tutor foi definida com atributos como Id, Nome, Telefone e Email, enquanto a entidade Pet possui Id, Nome, Raça, Espécie e TutorId, que serve como chave estrangeira, estabelecendo a ligação com a entidade Tutor. Para garantir a consistência dos dados inseridos, foram utilizadas anotações de validação como [Required] e [MaxLength] diretamente nas classes de modelo. Isso assegura que os dados sejam verificados ainda na etapa de entrada, evitando erros na persistência.

Outro aspecto fundamental da aplicação é a utilização do SQLite como banco de dados, escolhido por sua leveza e facilidade de configuração, sendo ideal para testes e

desenvolvimento de aplicações de pequeno e médio porte. A configuração do banco foi feita por meio do arquivo appsettings.json, onde foi definida a string de conexão, e sua inicialização ocorreu através da classe AppDbContext, que herda de DbContext e representa o contexto da aplicação para o Entity Framework. Adicionalmente, foi realizada a configuração da injeção de dependência no arquivo Program.cs, onde os serviços e repositórios foram registrados no contêiner de injeção. Isso permite que as dependências sejam resolvidas automaticamente pelo framework, facilitando a manutenção e os testes da aplicação.

Durante o processo de desenvolvimento, os principais desafios enfrentados foram o correto mapeamento das entidades e a separação das responsabilidades entre as camadas. Em especial, garantir que cada camada se mantivesse independente e que houvesse uma comunicação clara e segura entre elas exigiu atenção aos princípios de design de software.

Conclui-se, portanto, que a adoção de uma arquitetura em camadas com o uso de padrões como Repository e Service, junto ao Entity Framework Core e à injeção de dependência, proporciona uma base para aplicações Web API. Essa abordagem favorece o sistema, além de promover boas práticas de desenvolvimento.