# Desenvolvimento de uma Web API

## Gerenciamento de Pedidos e Fornecedores

Bernardo Negrini Borba

Este trabalho tem como objetivo apresentar as boas práticas de desenvolvimento de APIs RESTful e exemplificar sua aplicação em um sistema de gerenciamento de pedidos e fornecedores, implementado em .NET 8 utilizando C#.

O projeto desenvolvido inclui operações CRUD (Create, Read, Update, Delete) para as entidades Pedido e Fornecedor, garantindo persistência de dados com o Entity Framework Core e uso do padrão Repository. Além disso, a injeção de dependência foi empregada para desacoplar as responsabilidades e promover a modularidade do sistema.

Um dos pilares para o desenvolvimento de APIs RESTful é o design intuitivo de endpoints. Os recursos devem ser identificados por URLs claras e consistentes, e as operações devem ser mapeadas para os verbos HTTP apropriados. Na API desenvolvida, os endpoints seguem essas diretrizes:

1. GET /api/pedidos: Retorna todos os pedidos cadastrados.
2. POST /api/pedidos: Adiciona um novo pedido.
3. PUT /api/pedidos/{id}: Atualiza os dados de um pedido específico.
4. DELETE /api/pedidos/{id}: Remove um pedido.

A utilização de códigos de status HTTP para refletir o resultado das operações garante que os clientes entendam as respostas do servidor. Por exemplo, a criação de um pedido retorna 201 Created, enquanto a tentativa de acessar um pedido inexistente retorna 404 Not Found.

A organização modular do código facilita a manutenção e evolução do sistema. A aplicação foi estruturada em pastas com responsabilidades bem definidas:

Controllers: Contêm a lógica de interação com os clientes, implementando os endpoints da API.

Repositories: Abstraem o acesso aos dados, centralizando as operações de persistência.

Models: Definem as entidades do sistema, como Pedido e Fornecedor.

Data: Gerencia o contexto do banco de dados e as configurações do Entity Framework Core.

Essa divisão permite que cada componente seja desenvolvido, testado e alterado de forma independente, promovendo o princípio da separação de responsabilidades.

O padrão Repository foi utilizado para isolar a lógica de acesso ao banco de dados. Uma interface genérica, IRepository<T>, foi criada para encapsular operações comuns, como listar, buscar, adicionar, atualizar e remover entidades:

```
public interface IRepository<T> where T : class
{
    Task<IEnumerable<T>> GetAllAsync();
    Task<T?> GetByIdAsync(int id);
    Task AddAsync(T entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(int id);
}
```

Com essa abordagem, foi possível implementar um repositório reutilizável para as entidades Pedido e Fornecedor, simplificando a manutenção do código e possibilitando testes unitários mais robustos.

A injeção de dependência foi configurada para gerenciar as instâncias dos repositórios e o contexto do banco de dados. No arquivo Program.cs, o container de dependências foi configurado da seguinte forma:

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddScoped(typeof(IRepository<>), typeof(Repository<>));
```

Essa prática elimina o acoplamento entre os controladores e as implementações dos repositórios, promovendo a reutilização e a flexibilidade no código.

A validação de dados foi implementada diretamente nas classes de modelo, utilizando anotações como [Required] e [StringLength]. Isso garante que os dados enviados pelos clientes estejam no formato esperado antes de serem processados. Além disso, a documentação foi gerada automaticamente com o Swagger, oferecendo uma interface interativa para explorar os endpoints da API.

O desenvolvimento de APIs RESTful exige a aplicação de boas práticas que assegurem a qualidade do código e a satisfação dos requisitos funcionais. Nesta aplicação, as diretrizes foram seguidas por meio da padronização de endpoints, organização modular do código, uso do padrão Repository e injeção de dependência.

Essas práticas não apenas garantiram o cumprimento dos requisitos do sistema, mas também prepararam a aplicação para evoluções futuras. O projeto demonstra como a adoção de boas práticas no desenvolvimento de APIs pode transformar conceitos teóricos em soluções práticas, eficientes e escaláveis.