Desenvolvimento de uma Web API em .NET 8 para Gerenciamento de Pedidos e Fornecedores

As APIs RESTful são fundamentais nos sistemas modernos, permitindo que diferentes aplicações se conectem e troquem informações. Neste projeto, foi desenvolvida uma Web API em .NET 8 para gerenciar pedidos e fornecedores, com operações CRUD (Criar, Consultar, Atualizar, Excluir). Além disso, aplicamos boas práticas de desenvolvimento, como uso do Entity Framework Core, padrão Repository e injeção de dependência.

A API foi feita para lidar com dois domínios separados, pedidos e fornecedores, com funcionalidades independentes. Cada entidade possui seus próprios endpoints, o que deixa a aplicação mais flexível e fácil de expandir. Usamos o banco de dados SQLite para armazenar as informações, garantindo simplicidade e portabilidade.

Arquitetura e Tecnologias Usadas

O projeto foi organizado de forma modular, separando as responsabilidades em pastas específicas. As principais pastas e suas funções são:

- Models: Contém as classes das entidades (Pedido e Fornecedor), com as propriedades que representam os dados.
- Data: Inclui a classe AppDbContext, que gerencia a conexão com o banco e mapeia as entidades.
- Repositories: Implementa o padrão Repository, centralizando a lógica de acesso aos dados.
- Controllers: Contém os endpoints que expõem as operações CRUD para os usuários.

A API trabalha com duas entidades principais: Pedido e Fornecedor.

Pedido

As propriedades do pedido são: Id, Data, ValorTotal, Status e Descricao. Os endpoints para gerenciar pedidos são:

- GET /api/pedidos: Lista todos os pedidos.
- GET /api/pedidos/{id}: Retorna os detalhes de um pedido específico.
- POST /api/pedidos: Cria um novo pedido.
- PUT /api/pedidos/{id}: Atualiza um pedido existente.
- DELETE /api/pedidos/{id}: Remove um pedido do sistema.

Fornecedor

As propriedades do fornecedor são: Id, Nome, CNPJ, Telefone, Email e Endereco. Os endpoints são:

- GET /api/fornecedores: Lista todos os fornecedores.
- GET /api/fornecedores/{id}: Mostra os detalhes de um fornecedor.
- POST /api/fornecedores: Adiciona um novo fornecedor.
- PUT /api/fornecedores/{id}: Atualiza as informações de um fornecedor.
- DELETE /api/fornecedores/{id}: Remove um fornecedor do sistema.

Todos os endpoints seguem os princípios RESTful, garantindo clareza e padrão na comunicação.

Padrão Repository

Para separar a lógica de acesso aos dados da lógica de negócio, usamos o padrão Repository. Ele cria uma camada intermediária que centraliza os métodos de manipulação de dados. Por exemplo, a interface IPedidoRepository ficou assim:

csharp

Copiar código

```
public interface IPedidoRepository
{
    IEnumerable<Pedido> GetAll();
    Pedido GetById(int id);
    void Add(Pedido pedido);
    void Update(Pedido pedido);
    void Delete(int id);
}
```

Essa abordagem deixa o código mais organizado, reutilizável e fácil de testar.

Persistência de Dados

O Entity Framework Core gerencia o acesso ao banco de dados. Criamos o banco de dados com as ferramentas de migração do EF Core. Depois de definir as entidades, executamos os comandos:

bash

Copiar código

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Isso garantiu que o banco estivesse alinhado com o código do projeto.

Injeção de Dependência

Usamos a injeção de dependência para gerenciar os repositórios e o contexto do banco de dados. Configuramos tudo no Program.cs:

csharp

Copiar código

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite("Data Source=gerenciamento.db"));
builder.Services.AddScoped<IPedidoRepository, PedidoRepository>();
builder.Services.AddScoped<IFornecedorRepository,
FornecedorRepository>();
```

Com isso, o framework cria e gerencia as instâncias automaticamente.

Boas Práticas

Durante o desenvolvimento, seguimos algumas boas práticas:

- Organização Modular: O projeto foi dividido em pastas claras para facilitar a navegação e manutenção.
- Nomenclatura Descritiva: Nomes de variáveis, métodos e classes foram pensados para serem fáceis de entender.
- Entity Framework Core: Automatizamos o acesso ao banco de dados, reduzindo erros.
- RESTful: Garantimos que a API seguisse os padrões REST, facilitando o uso e integração.

Essa abordagem resultou em uma API bem estruturada, eficiente e fácil de expandir.

Conclusão

O desenvolvimento desta Web API em .NET 8 demonstrou como aplicar os conceitos fundamentais de APIs RESTful, padrões de design e boas práticas de programação. A utilização do padrão Repository, injeção de dependência e Entity Framework Core tornou o sistema robusto, escalável e fácil de manter. A aplicação está pronta para atender às necessidades de gerenciamento de pedidos e fornecedores, servindo como base para futuras expansões e integrações. Este projeto reforça a importância de práticas bem estabelecidas no desenvolvimento de software, garantindo a entrega de um produto de qualidade.