O desenvolvimento de APIs RESTful tornou-se um padrão adotado no desenvolvimento de aplicações modernas, devido à sua simplicidade, flexibilidade e escalabilidade. APIs RESTful permitem a comunicação entre diferentes sistemas, facilitando a integração de serviços e possibilitando o compartilhamento de recursos de maneira eficiente. As boas práticas no desenvolvimento de APIs RESTful são essenciais para garantir que a aplicação seja de fácil manutenção, escalável e segura.

Uma das principais boas práticas em uma API RESTful é o uso adequado dos métodos HTTP. Cada método HTTP possui um propósito específico e deve ser utilizado de forma apropriada para garantir a clareza da API. O método GET deve ser usado para obter informações, o método POST para criar novos recursos, o método PUT para atualizar recursos existentes e o método DELETE para remover recursos. Na aplicação desenvolvida, essas práticas são claramente seguidas nas rotas para os recursos de pedido e fornecedor. Por exemplo, no código da classe PedidoController, o método GetAll utiliza o método GET para retornar todos os pedidos, enquanto o método Post usa o POST para criar um novo pedido. Da mesma forma, os métodos Put e Delete são utilizados para atualizar e excluir um pedido, respectivamente.

Na aplicação desenvolvida, as rotas são definidas de maneira clara e seguem um padrão lógico. Por exemplo, a rota GET /pedidos retorna todos os pedidos, enquanto GET /pedidos/{id} retorna um pedido específico. Isso garante que a API seja fácil de navegar e entender, seguindo as convenções REST. Além disso, o uso de convenções RESTful na nomenclatura das rotas torna a API mais intuitiva, permitindo que desenvolvedores compreendam rapidamente os recursos disponíveis.

A utilização de um banco de dados relacional com o Entity Framework Core é uma outra prática comum e eficaz no desenvolvimento de APIs RESTful. O uso do Entity Framework Core facilita o acesso e a manipulação de dados, abstraindo detalhes de implementação de banco de dados e tornando o código mais limpo e legível. Na aplicação, as classes AppDbContext e AppDbContext2 são responsáveis pela configuração do contexto do banco de dados, permitindo que os dados dos pedidos e fornecedores sejam armazenados e acessados facilmente. O uso do DbSet<> nas classes AppDbContext e AppDbContext2 permite que as operações de criação, leitura, atualização e exclusão (CRUD) sejam realizadas de forma simples e eficiente.

Os controllers do código foram feitos para o pedido e para o fornecedor, usando comandos como [HttpGet] que retorna todos os pedidos. O [HttpGet("{id}")]

busca um pedido específico pelo ID. O [HttpPost] cria um novo pedido, enquanto o [HttpPut("{id}")] atualiza um pedido existente, se encontrado. O [HttpDelete("{id}")] exclui um pedido com base no ID fornecido. Cada método utiliza códigos de status HTTP para indicar o sucesso ou falha das operações: 200 OK, 404 Not Found, 201 Created e 204 No Content. Esse controlador segue as boas práticas RESTful, utilizando métodos HTTP apropriados para cada ação e garantindo respostas claras com base no resultado das requisições.

As classes Pedido e Fornecedor portam as propriedades que serão utilizadas em cada classe, onde o “Id” aparece em ambas, para que assim, seja possível identificá-las pelo seu código quando utilizarmos o Swagger, com o método Get{id}, onde este e outros métodos foram criados nas interfaces, tanto do fornecedor, quanto a do pedido, podendo atualizar, adicionar, deletar e pegar dados no banco de dados. Sendo na classe PedidoRepository, que herda a IPedidoRepository, onde estes métodos estão sendo utilizados e contém suas respectivas funções. O mesmo ocorre para a classe FornecedorRepository, que herda de IFornecedorRepository e possui os mesmos métodos de Get, Put, Post e Delete, sendo adicionado funções para estes métodos, assim como no PedidoRepository.