Introdução:
As APIs desempenham papel fundamental na comunicação entre diferentes sistemas e aplicações, uma abordagem muito utilizada é da API RESTful, que oferece flexibilidade e maior facilidade de manutenção, as APIs RESTful servem para desenvolver interfaces de programação, que sigam os princípios do modelo REST, as APIs RESTful aderem estritamente aos princípios REST, sendo consideradas uma implementação mais rigorosa deste modelo, para ser considerada RESTful ela deve seguir alguns requisitos a mais, sendo eles a Interface Uniforme, Clientes sem Estado e Operações baseadas em recursos.
A abordagem de API RESTful foi utilizada no projeto, visando facilitar a programação, utilizando de algumas boas práticas de desenvolvimento.

Desenvolvimento:
Para maior compreensão da utilização das boas práticas, as mesmas serão explicadas, elas consistem no uso correto dos verbos HTTP, sendo eles GET (Recupera informação de recurso), POST (Cria novo recurso), DELETE (Remove recurso existente) e PUT (Atualiza informação de recurso existente), as rotas devem ser claras, documentar a API é muito importante, Swagger e outras ferramentas semelhantes facilitam a visualização da documentação de forma interativa, a arquitetura do programa deve ser dividida em camadas bem definidas, como controladores, repositórios e modelos, facilitando em uma possível manutenção do programa.
No projeto, as boas práticas foram bem utilizadas, exemplificando uso de cada parte, os Verbos HTTP e Estruturação de rotas foram aplicados de forma clara:

```
[HttpGet]
public async Task<IActionResult> GetAll () =>
Ok(await _fornecedorRepository.GetAllAsync ());
```

Esta parte se localiza no FornecedorController.cs, também tendo utilização parecida em PedidosController.cs. A rota neste caso está estruturada corretamente, para /api/fornecedores.

Já quanto às respostas HTTP, quando procurado por um fornecedor utilizando o ID, é utilizado uma resposta apropriada:

```
[HttpGet(“{id}”)]
public async Task<IActionResult> GetById(int id)
{
var fornecedor = await
_fornecedorRepository.GetByIdAsync(id);
return fornecedor != null ? Ok(fornecedor) : NotFound();
}
```

No exemplo acima, localizado em FornecedorController.cs, ele busca pelo recurso, caso consiga, retornará o código 200 OK, caso não consiga, retornará o código 404 NotFound. Situação parecida também pode ser vista em PedidosController.cs.

Validação de dados também é vista no projeto, quando criado um novo pedido, os dados são validados primeiramente:

```
public async Task<IActionResult> Create(Pedido pedido)
{
if (pedido.ValorTotal <= 0)
return BadRequest (“Valor total deve ser maior que zero.”);
await _pedidoRepository.AddAsync (pedido);
return CreatedAtAction (nameof (GetById), new { id = pedido.Id }, pedido);
}
```

Esta parte do código se localiza em PedidosController.cs.

A ferramenta Swagger foi integrada no projeto, para fornecer uma interface gráfica:

```
builder.Services.AddSwaggerGen();
app.UseSwagger();
app.UseSwaggerUI();
```

A integração do Swagger pode ser vista no Program.cs, graças a ela as rotas podem ser testadas direto no navegador, o que ajuda na acessibilidade e na transparência do projeto.

A separação de camadas também foi uma das boas práticas vistas no projeto, o código foi dividido entre Models, Controllers e Repositories, visando maior facilidade na manutenção do projeto e ajudando na organização do mesmo.

Conclusão:
O projeto foi concluído utilizando de todas as boas práticas de desenvolvimento de APIs RESTful, resultando em um projeto organizado e de fácil manutenção, todo o conteúdo ensinado foi repassado para o código, resultando na entrega de um projeto completo e bem executado.

Fontes utilizadas:
https://www.dio.me/articles/entendendo-as-diferencas-entre-apis-rest-e-restful
https://www.hostinger.com.br/tutoriais/api-restful
https://www.brunobrito.net.br/api-restful-boas-praticas/