Objetivo do Teste:

Neste exercício, você será desafiado a implementar métodos CRUD (Criar, Ler, Atualizar e Excluir) para uma API em C# usando ASP.NET Core com SQLite. Cada método deverá ser implementado de acordo com requisitos específicos que envolvem manipulação de strings, validação de dados, uso de laços de repetição, ordenação de listas e tratamentos de exceções.

---

1. Criar Produto (POST)

Objetivo: Implementar o método para criar um produto na base de dados.

Instruções:

- O produto deve ter os seguintes atributos:
    - Id (auto gerado pelo banco de dados)
    - Nome (string, deve ser um nome descritivo do produto)
    - Preco (decimal, valor do preço do produto)
- **Validação**:
    - O nome não pode ser vazio ou nulo.
    - O preço deve ser maior que zero.
- **Manipulação de String**:
    - O nome do produto, caso seja informado em letras minúsculas, deve ser convertido para letras maiúsculas ao ser inserido no banco.

Exemplo de Endpoint:

- **POST /api/produtos**
    - Corpo da requisição:

Json

```
{
  "nome": "produto exemplo",
  "preco": 50.00
}
```

- **Resposta esperada:**
    - Retorne o produto criado com o nome convertido para maiúsculas:

Json

```
{
  "id": 1,
```

```
  "nome": "PRODUTO EXEMPLO",
  "preco": 50.00
}
```

---

2. Obter Produtos (GET)

Objetivo: Implementar o método para listar todos os produtos disponíveis.

Instruções:

- Recupere todos os produtos armazenados na base de dados.
- **Laço de Repetição**:
    - Percorra a lista de produtos e aplique um filtro de exemplo (por exemplo, se o nome do produto contiver a palavra "promoção", adicione um marcador no nome do produto como "[Em Promoção]").
- **Ordenação de Lista**:
    - Ordene os produtos pelo preço em ordem crescente (do mais barato para o mais caro).

Exemplo de Endpoint:

- **GET /api/produtos**

Resposta esperada:

```
json[
 {
   "id": 1,
   "nome": "PRODUTO EXEMPLO",
   "preco": 50.00
 },
 {
   "id": 2,
   "nome": "PRODUTO EM PROMOÇÃO",
   "preco": 30.00
 }
]
```

---

3. Atualizar Produto (PUT)

Objetivo: Implementar o método para atualizar os dados de um produto existente.

Instruções:

- O produto a ser atualizado pode ter os seguintes atributos:
    - Nome (string)
    - Preco (decimal)
- **Validação**:
    - Verifique se o produto existe no banco de dados. Caso contrário, retorne um erro 404 - Not Found.
    - O preço não pode ser menor que zero.
- **Manipulação de String**:
    - Se o nome for alterado, altere a primeira letra do nome para maiúscula.

Exemplo de Endpoint:

- **PUT /api/produtos/{id}**
    - Corpo da requisição:

Json

```
{
  "nome": "produto novo",
  "preco": 80.00
}
```

Resposta esperada:

json

```
{  "id": 1,
   "nome": "Produto Novo",
    "preco": 80.00
}
```

---

4. Excluir Produto (DELETE)

Objetivo: Implementar o método para excluir um produto da base de dados.

Instruções:

- O produto será identificado pelo Id.
- Verifique se o produto com o Id fornecido existe. Caso contrário, retorne um erro 404 - Not Found.

- **Exclusão de Produto**:
    - Após excluir o produto, retorne um status 204 - No Content.

Exemplo de Endpoint:

- **DELETE /api/produtos/{id}**

Resposta esperada:

    - Retorne apenas um status de sucesso 204.

---

5. Validação de Dados em Criar e Atualizar Produto

Objetivo: Adicionar validação de dados para garantir que as informações enviadas são válidas.

Instruções:

- O nome do produto deve ser uma string não vazia. Caso contrário, o método deve retornar um erro 400 - Bad Request.
- O preço do produto deve ser maior que zero. Caso contrário, o método deve retornar um erro 400 - Bad Request.
- **Exemplo de Validação**:
    - Nome vazio: "nome": "" → Retornar 400 - Bad Request.
    - Preço negativo: "preco": -10.00 → Retornar 400 - Bad Request.

Exemplo de Resposta de Erro:

Json

```
{
  "error": "O preço deve ser maior que zero."
}
```

---

6. Filtragem e Ordenação Avançada (Bônus)

Objetivo: Implementar a capacidade de filtrar e ordenar a lista de produtos com base no nome e preço.

Instruções:

- Adicione uma funcionalidade onde o usuário pode filtrar os produtos por nome (usando uma query string, como ?nome=produto).
- Ordene os produtos por preço em ordem crescente ou decrescente, dependendo do parâmetro ?ordem=asc ou ?ordem=desc.

Exemplo de Endpoint:

- **GET /api/produtos?nome=produto&ordem=desc**

Resposta esperada:

json

```
[
 {
  "id": 2,
  "nome": "PRODUTO CARO",
  "preco": 200.00
 },
 {
  "id": 1,
  "nome": "PRODUTO EXEMPLO",
  "preco": 50.00
 }
]
```

Instruções Adicionais:

- **Tecnologias**: Utilize **ASP.NET Core** para criar a API e **Entity Framework** com **SQLite** para persistência de dados.
- **Teste da API**: Após a implementação, use o Swagger ou o Postman para testar os endpoints.
- **Estrutura do Projeto**: O projeto deve ser estruturado com boas práticas de design, utilizando controllers, models e contexto de banco separados.
- **Documentação**: Documente todos os endpoints, incluindo parâmetros de entrada e formato de resposta, usando comentários no código ou uma documentação adicional.

Critérios de Avaliação:

- Implementação correta dos métodos CRUD.
- Validação de dados e tratamento de erros adequado.
- Uso de manipulação de strings e boas práticas de código.
- Uso de laços de repetição e ordenação de listas.
- Clareza no código e organização do projeto.

- Capacidade de testar e garantir o funcionamento dos endpoints.