

|  |  |  |
| --- | --- | --- |
| **Sprint Documentation** | | |
| **Sprint #07** | **Start Date: 02/06/2025** | **Final Date: 06/06/2025** | |
| **Team Members** | | | |
| Renato dos Santos Filho | | | |

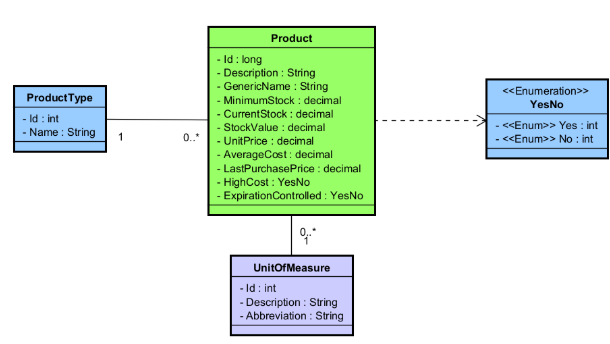
|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint Backlog** | | | |
| Task# | Description | Start Date | Final date |
| 89 | Sprint 07 - Desenvolver a API para Product | 02/06/2025 | 06/06/2025 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task Description** | | | | | |
| Task # | Description | Assigned To | Status | Estimated Hours | Logged Hours |
| 89 | Sprint 07 - Desenvolver a API para Product | Renato Santos | Concluído | 8 | 8 |

|  |
| --- |
| **Sprint Results** |

Foi desenvolvido uma API para a classe Product. Esse processo envolveu a criação de diversos arquivos nas diferentes camadas do sistema para que as funções executassem corretamente.

Primeiramente foram criados a model e o DTO conforme a seguinte representação no diagrama de classes:



Os atributos são:

 Id, do tipo long, representa a chave primária dos registros nessa classe.

 Description, do tipo string, Varchar(255) no banco de dados, representa a descrição do produto.

 GenericName, do tipo string, Varchar(100) no banco de dados, representa o nome genérico do produto.

 MinimumStock, do tipo decimal, representa o estoque mínimo permitido para o produto.

 CurrentStock, do tipo decimal, representa a quantidade atual em estoque do produto.

 StockValue, do tipo decimal, representa o valor total atual do estoque do produto.

 UnitPrice, do tipo decimal, representa o preço unitário do produto.

 AverageCost, do tipo decimal, representa o custo médio do produto.

 LastPurchasePrice, do tipo decimal, representa o preço da última compra realizada do produto.

 HighCost, do tipo YesNo (enum ou char(1)), indica se o produto é de alto custo (Y ou N).

 ExpirationControlled, do tipo YesNo (enum ou char(1)), indica se o produto é controlado por validade (Y ou N).

Em seguida, foram criadas as camadas Data, Service e a Controller, e inseridos os seguintes registros como carga inicial de dados para a classe.

Na controller, os seguintes endpoints são aceitos:

**HttpGet**: para buscar todos os registros da classe.

**HttpGet com id**: para buscar um registro a partir do id fornecido.

**HttpPost**: para criar um novo registro no banco de dados.

**HttpPut**: para atualizar um registro a partir do id e dos dados fornecidos.

**HttpDelete**: para remover um registro existente a partir do id passado na url da requisição.

Durante o desenvolvimento, foram adicionadas validações para garantir a integridade e consistência dos dados:

* Validação de dados obrigatórios e formatos corretos:
* Impedida a criação ou atualização de registros com:
* Emails em formato inválido
* CPF/CNPJ fora do padrão ou inválidos
* Telefones com formatação incorreta
* Prevenção de registros duplicados:

Antes de inserir um novo registro, é feita a verificação de unicidade com base nos campos necessários (como nome, código ou identificadores específicos).

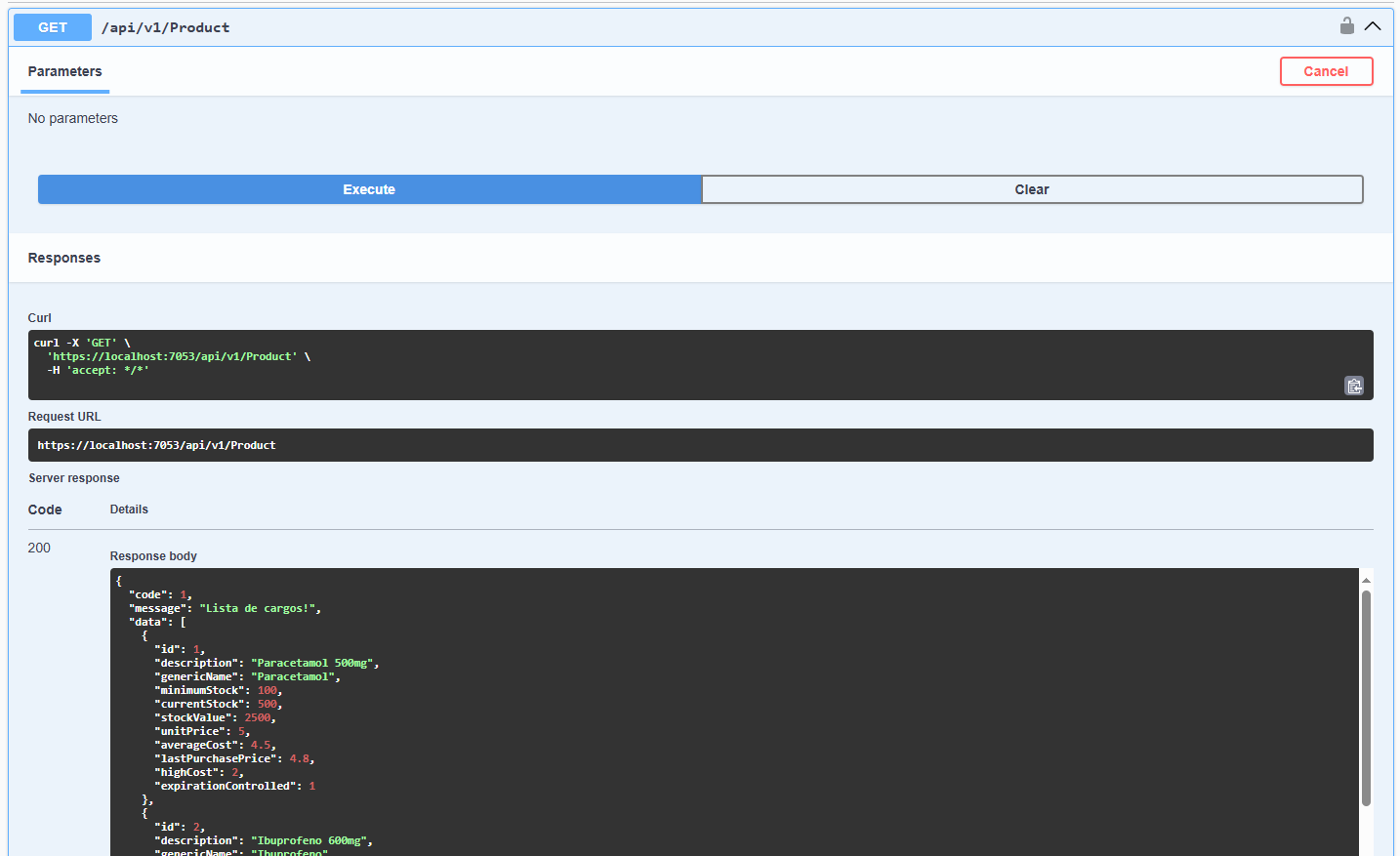
Resposta com 409 Conflict caso um registro semelhante já exista.

Validação na exclusão de dados:

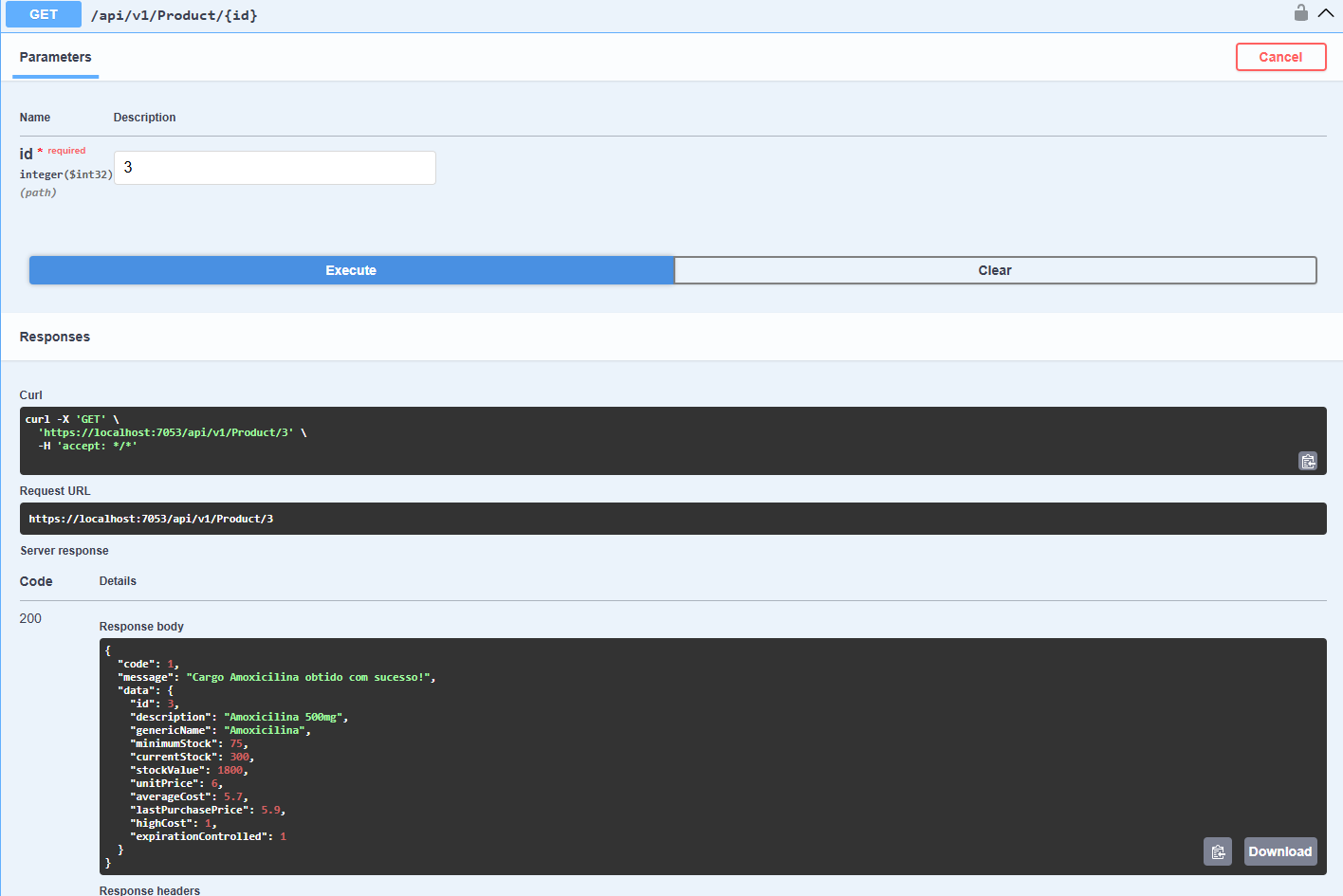
Ao chamar o método DELETE, é feita uma verificação se o produto está relacionado a registros em entidades dependentes (relação um-para-muitos).

A exclusão só é permitida caso não haja dependentes relacionados ao produto.

Telas dos resultados:

Endpoint GET: 

Endpoint POST com validação: 

Endpoint GET BY ID com validação: 

Endpoint PUT com validação:  


Endpoint delete com validação:

