**IT Happens**

**Avaliação Técnica**

Prova: **it-happens-1303**  
Desafio Técnico: **test-ithappens-1304**

Nome do Projeto: **MateusStore**  
Programador Responsável: **Antonio Alan Menezes da Silva**Linguagem de Programação: **C#**Mecanismo de Banco de Dados: **SQL Server**  
Nome do Banco de Dados: **MateusStore**

Sumário

[COMPETÊNCIA DESENVOLVIMENTO 3](#_Toc28014533)

[Metodologias e Padrões Adotados 3](#_Toc28014534)

[Observações 3](#_Toc28014535)

[COMPETÊNCIAS DE BANCO DE DADOS 4](#_Toc28014537)

[Script de Criação do Banco de Dados 4](#_Toc28014538)

[Diagrama Entidade-Relacionamento 4](#_Toc28014539)

[Consulta 01: Buscando produtos com quantidade maior ou igual a 100 5](#_Toc28014540)

[Consulta 02: Buscando produtos com estoque da Filial 60 5](#_Toc28014541)

[Consulta 03: Buscando todos os campos do PedidoEstoque e ItensPedido filtrado pelo produto de código 7993 5](#_Toc28014542)

[Consulta 04: Buscando todos os pedidos com suas respectivas formas de pagamento 5](#_Toc28014543)

[Consulta 05: Comparando se a soma dos valores dos itens de pedido bate com o valor total da capa do pedido 5](#_Toc28014544)

[Consulta 06: Buscando todos os pedidos com mais de 10 itens 5](#_Toc28014545)

Opções e [Validações da API 6](#_Toc28014546)

Opções Disponíveis na API [7](#_Toc28014547)

Gravando e Buscando Cliente [7](#_Toc28014547)

[Gravando e Buscando Filial **.**](#_Toc28014548)

[Gravando e Buscando Usuário **.**](#_Toc28014549)

[Gravando e Buscando Produto **.**](#_Toc28014550)

[Realizando Pedido de Entrada **.**](#_Toc28014551)

[Realizando Pedido de Saída (Venda) **.**](#_Toc28014552)

# COMPETÊNCIA DESENVOLVIMENTO

## Metodologias e Padrões Adotados

No desenvolvimento deste projeto foram adotados alguns padrões considerados boas práticas de programação:

* Buscou-se manter o **código simples**, sem necessidade de comentários explicativos, ou seja, auto documentado;
* Os construtores de classes foram mantidos privados e, para validação de dados, adotou-se o Método de Fábrica "Criar", aumentando a **coesão das classes**;
* Os métodos foram separados de forma que cada um realize apenas uma tarefa, respeitando assim o **princípio da responsabilidade única**;
* Os métodos de fábrica e alteradores de classes utilizam exceção para encerrar imediatamente a aplicação caso a classe se torne inválida, respeitando, assim, o **princípio fail fast**;
* O mini ORM **Dapper**, foi utilizado para a **camada de Persistência**, o qual permitiu um balanceamento adequado entre liberdade para escrever e praticidade nas requisições SQL;
* Para a **API**, fez-se uso de objetos de transferência de dados (**DTO**), **protegendo assim o domínio** do meio externo;
* A ferramenta **Swagger** foi utilizada para executar os testes por **requisições HTTP** à API;

### Observações

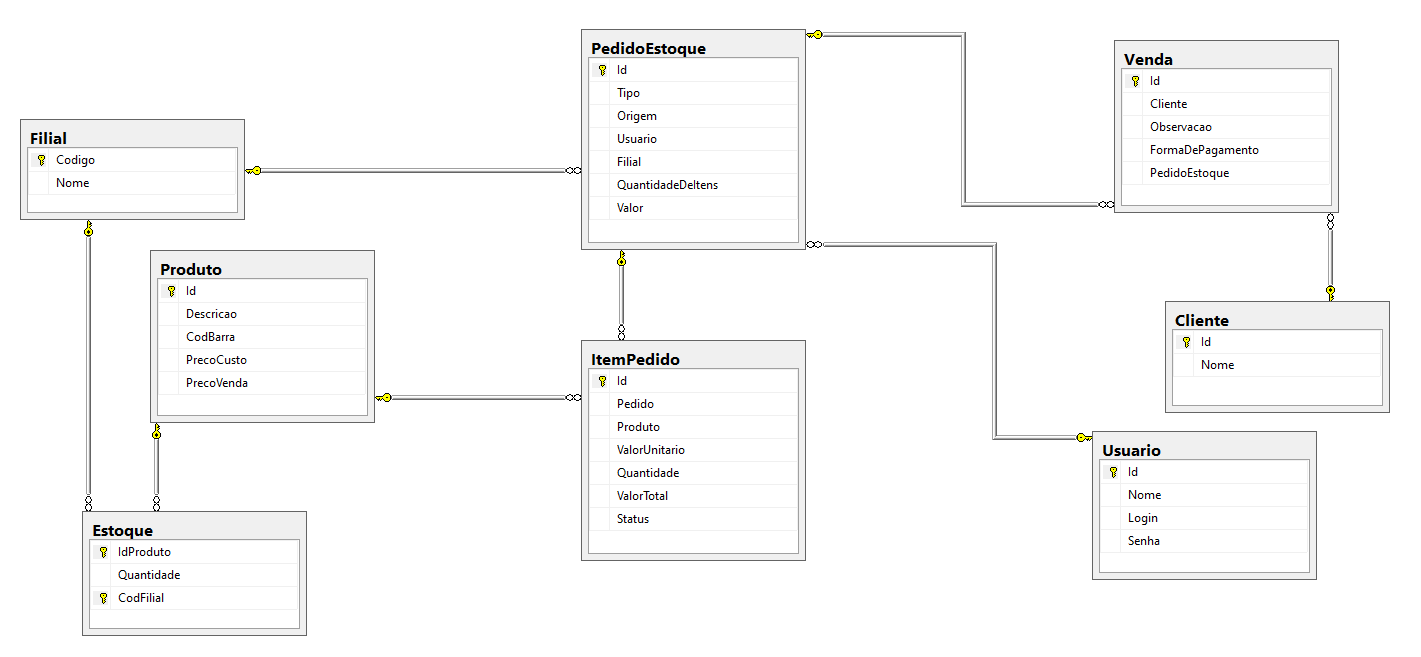
* O **Swagger** funciona de forma integrada com Visual Studio, bastando adicionar ao projeto via NuGet Packages (buscar por swashbuckle.aspnetcore);
* Na pasta de implementação do projeto há uma **pasta DB** que contém o **script** necessário para **criação do banco de dados** da aplicação.

# COMPETÊNCIAS DE BANCO DE DADOS

## Script de Criação do Banco de Dados

Disponível em: implementacao\BD \ScriptCreateDB.sql

## Diagrama Entidade-Relacionamento

  
Criado com o SQL Server Management

### Consulta 01: Buscando produtos com quantidade maior ou igual a 100

SELECT P.Descricao AS Produto

,P.CodBarra

,SUM (E.Quantidade) AS Quantidade

FROM Produto AS P

INNER JOIN Estoque AS E

ON (P.Id = E.IdProduto)

GROUP BY P.Descricao

,P.CodBarra

HAVING SUM (E.Quantidade) >= 100

### 

### Consulta 02: Buscando produtos com estoque da Filial 60;

SELECT P.Descricao AS Produto

,E.Quantidade

FROM Produto AS P

INNER JOIN Estoque AS E

ON (P.Id = E.IdProduto)

WHERE E.Quantidade > 0

AND E.CodFilial = 60

### Consulta 03: Buscando todos os campos do PedidoEstoque e ItensPedido filtrado pelo produto de código 7993

SELECT \*

FROM PedidoEstoque AS P

INNER JOIN ItemPedido AS I

ON (P.Id = I.Pedido)

WHERE I.Produto = 7993

### Consulta 04: Buscando todos os pedidos com suas respectivas formas de pagamento

SELECT P.\*

,CASE

WHEN V.FormaDePagamento = 1 THEN

'Avista'

WHEN V.FormaDePagamento = 2 THEN

'Boleto'

WHEN V.FormaDePagamento = 3 THEN

'Cartão'

END AS FormaDePagamento

FROM PedidoEstoque AS P

INNER JOIN Venda AS V

ON (P.Id = V.PedidoEstoque)

\*Obs: No modelo atual, só há forma de pagamento para vendas. Por isso o INNER JOIN com Venda.

### Consulta 05: Comparando se a soma dos valores dos itens de pedido bate com o valor total da capa do pedido

SELECT I.Pedido

,P.Valor AS TotalCapa

,SUM(I.ValorTotal) AS TotalItens

,CASE

WHEN P.Valor = SUM(I.ValorTotal) THEN

'Sim'

ELSE

'Não'

END AS Bateu

FROM PedidoEstoque AS P

INNER JOIN ItemPedido AS I

ON (P.Id = I.Pedido)

GROUP BY I.Pedido

,P.Valor

### Consulta 06: Buscando todos os pedidos com mais de 10 itens

SELECT I.Pedido

,COUNT(\*) AS TotalDeItens

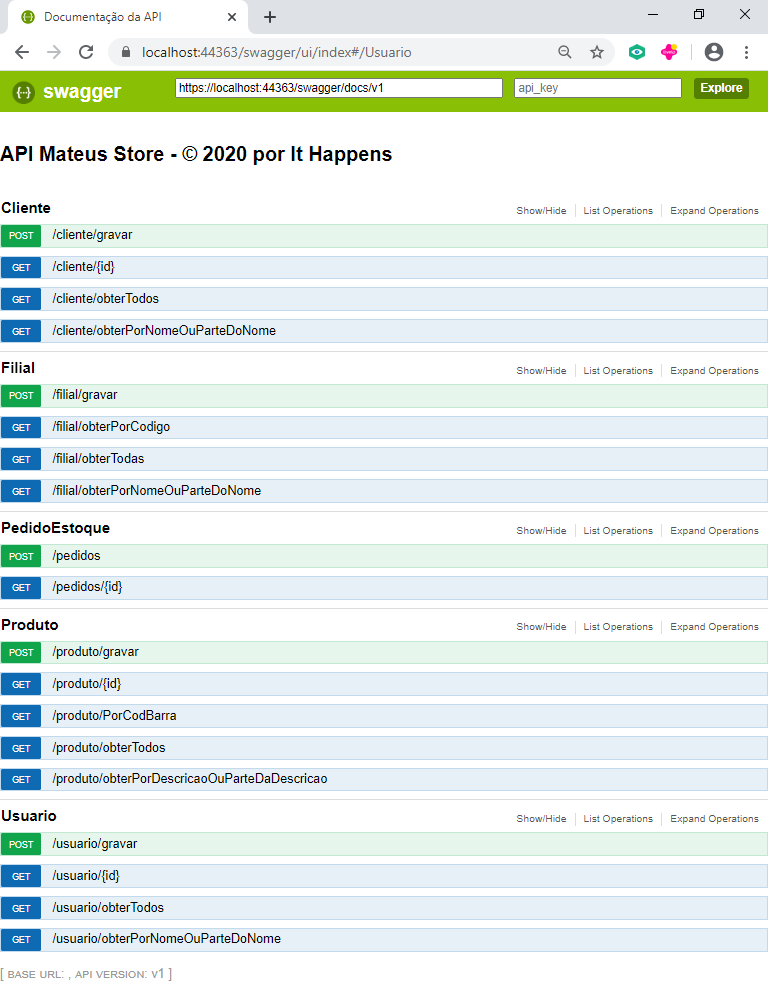
FROM ItemPedido AS I

GROUP BY I.Pedido

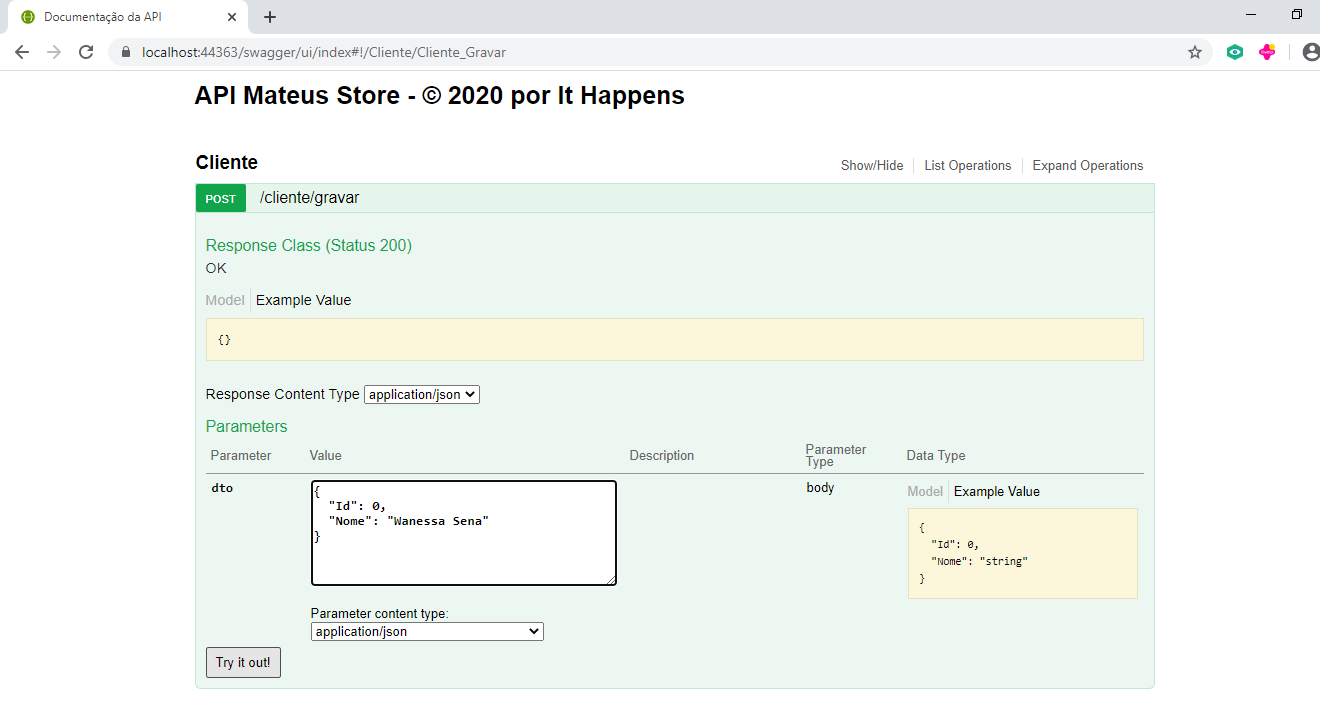
HAVING COUNT(\*) > 10

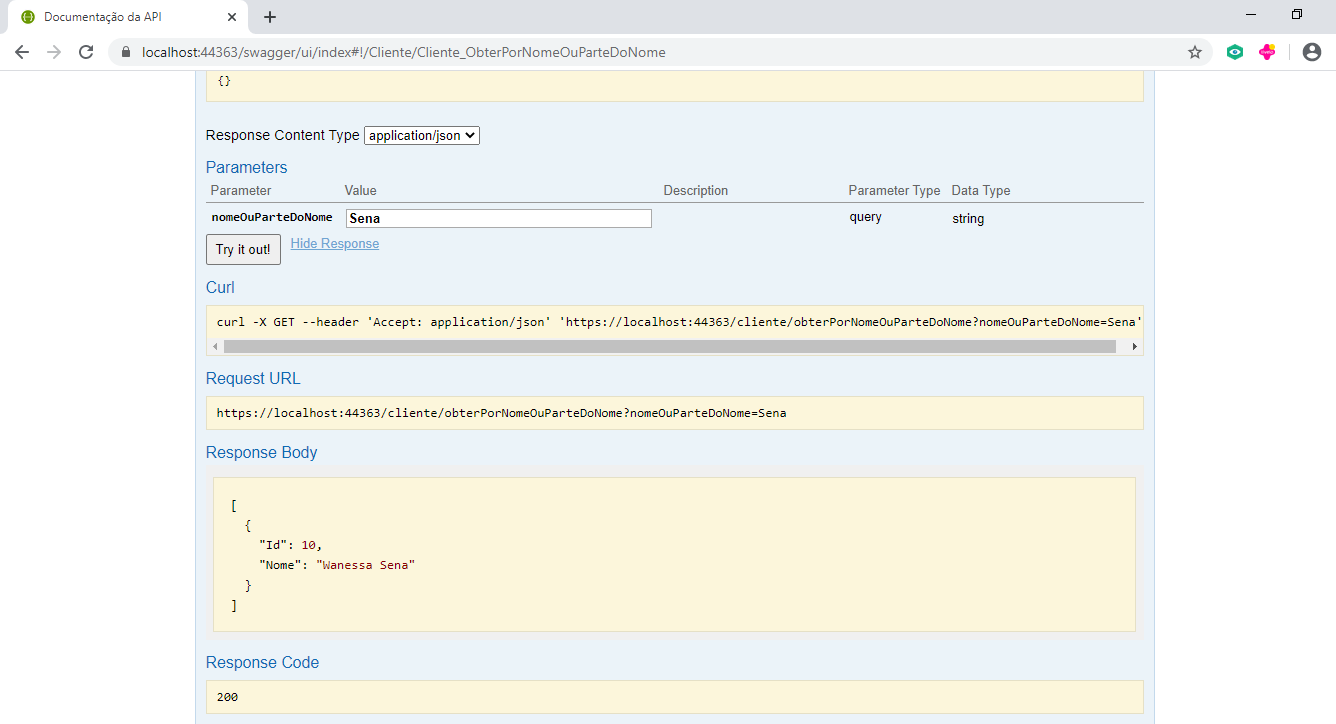
## Opções e Validações da API

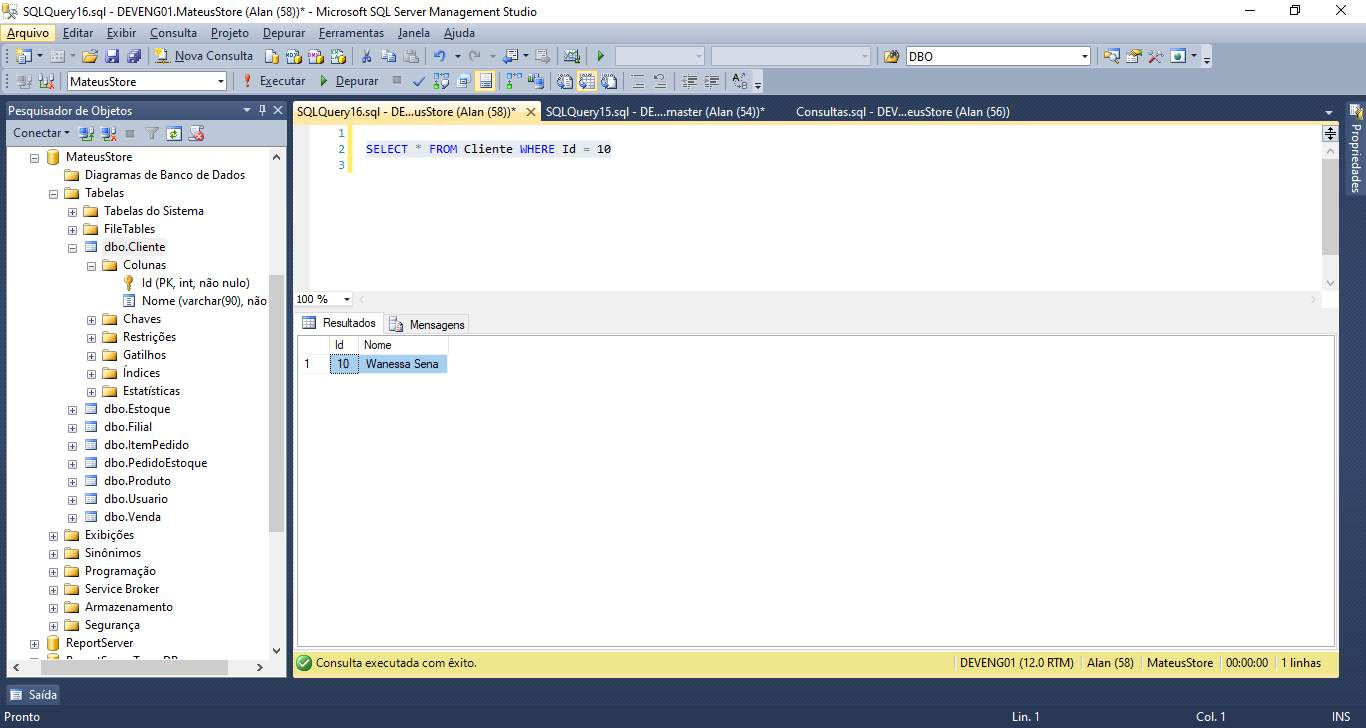
Opções Disponíveis na API



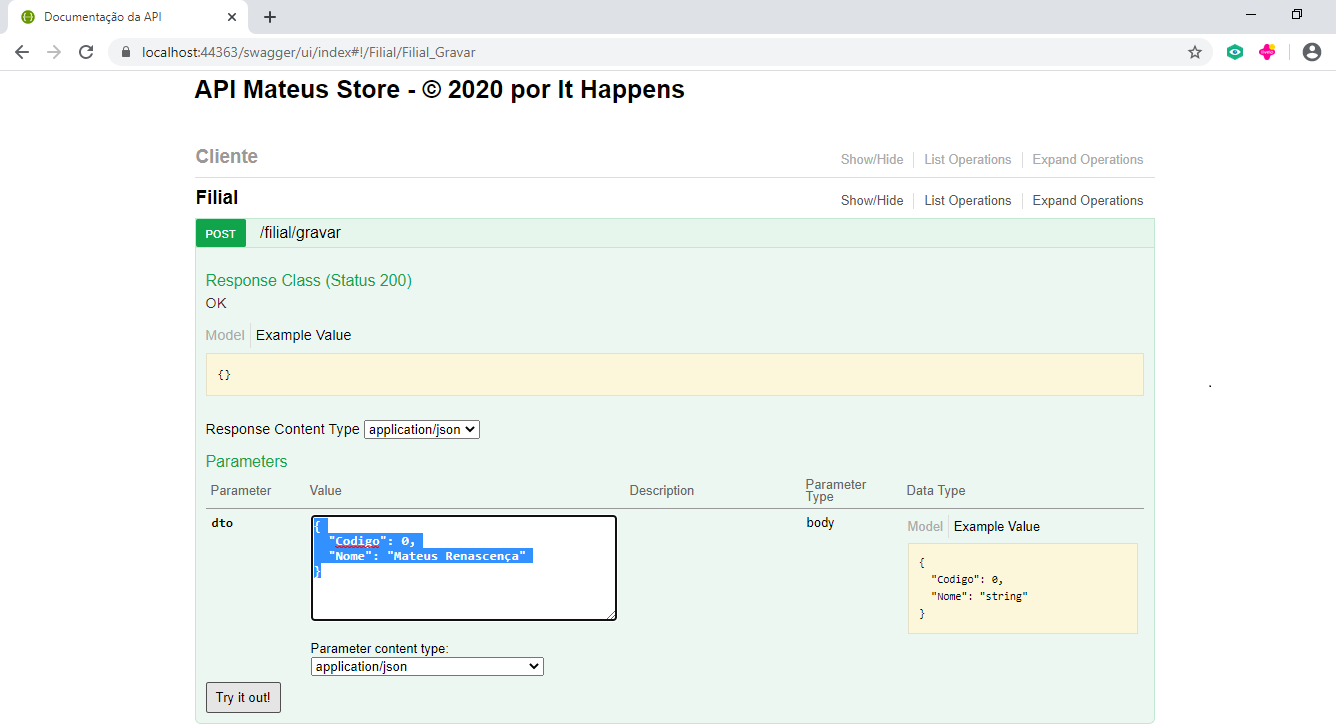
### Gravando e Buscando Cliente

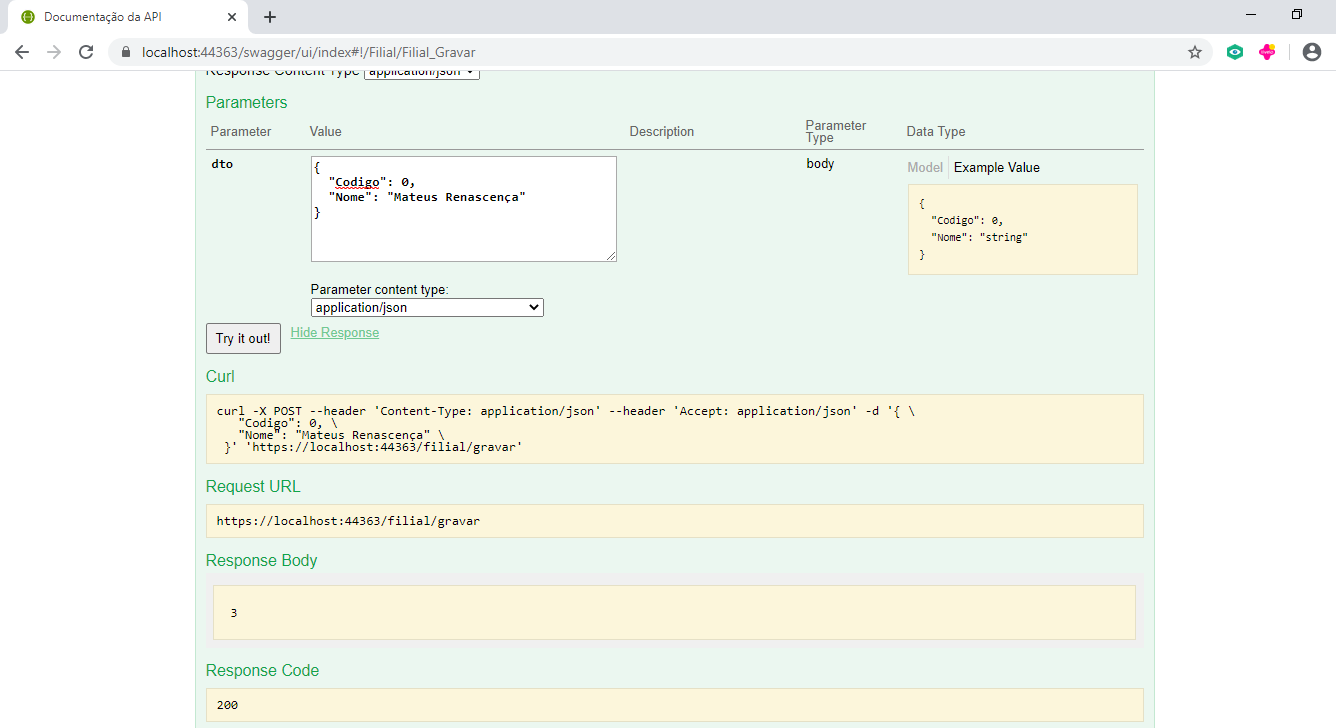


  
Resultado da requisição

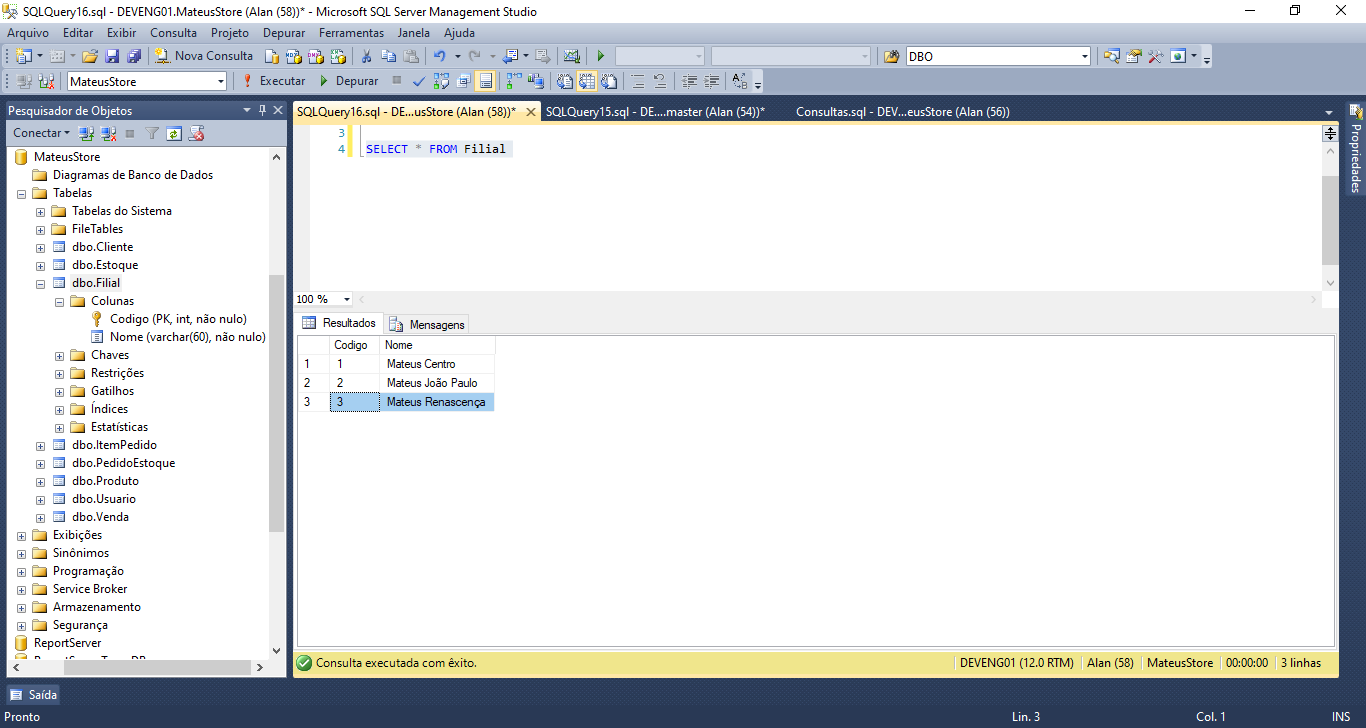
  
Verificando registro na base de dados

### Gravando e Buscando Filial

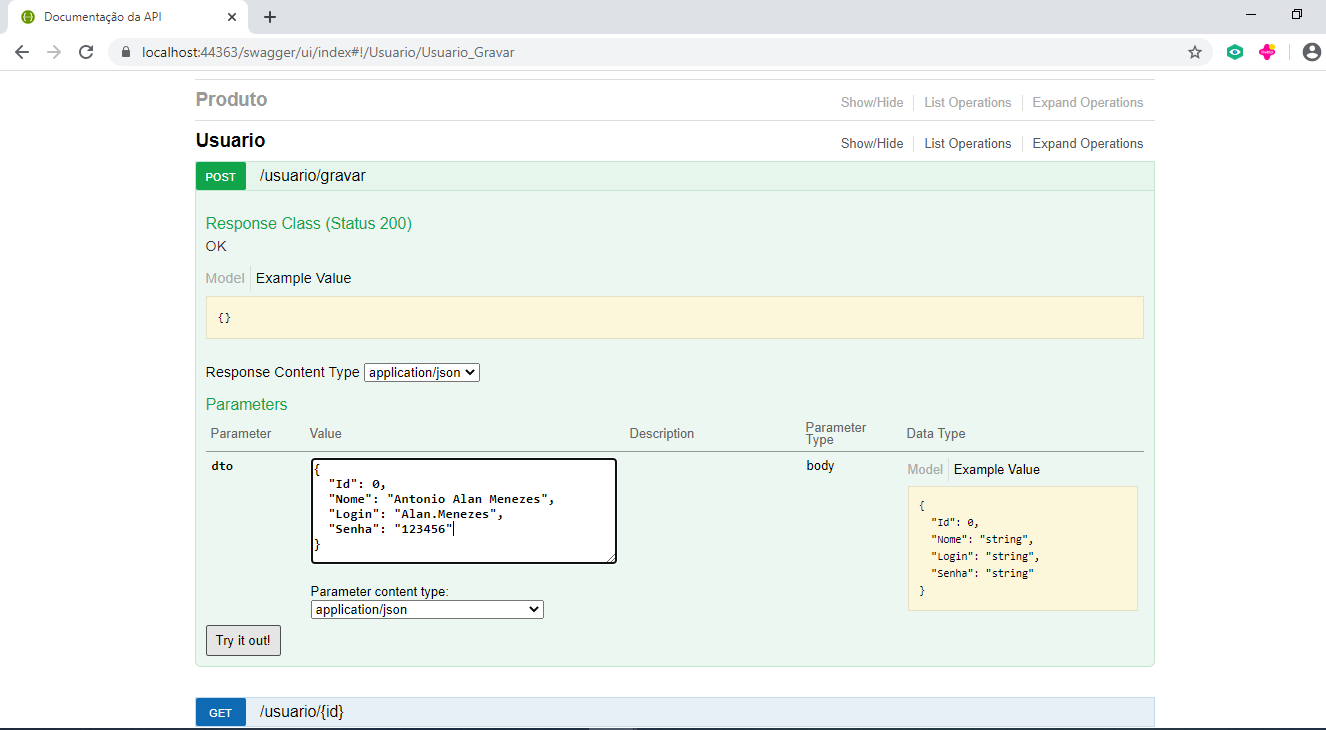


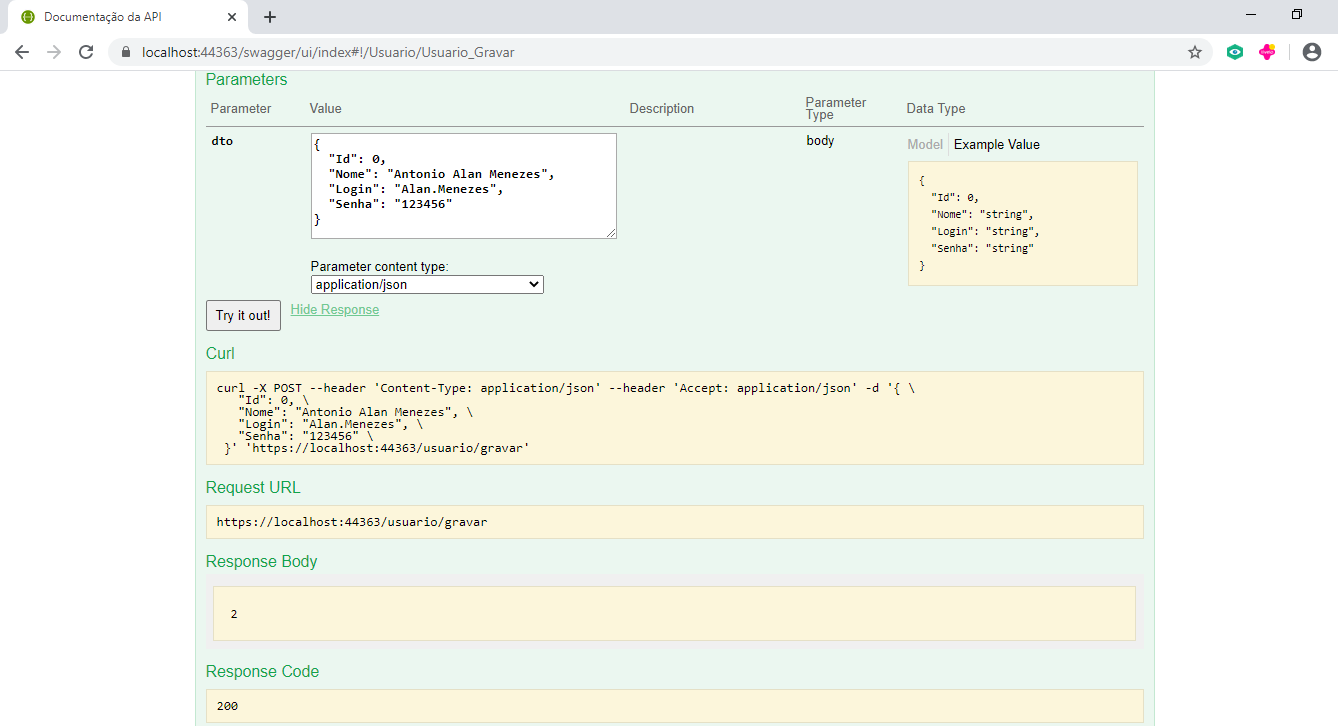


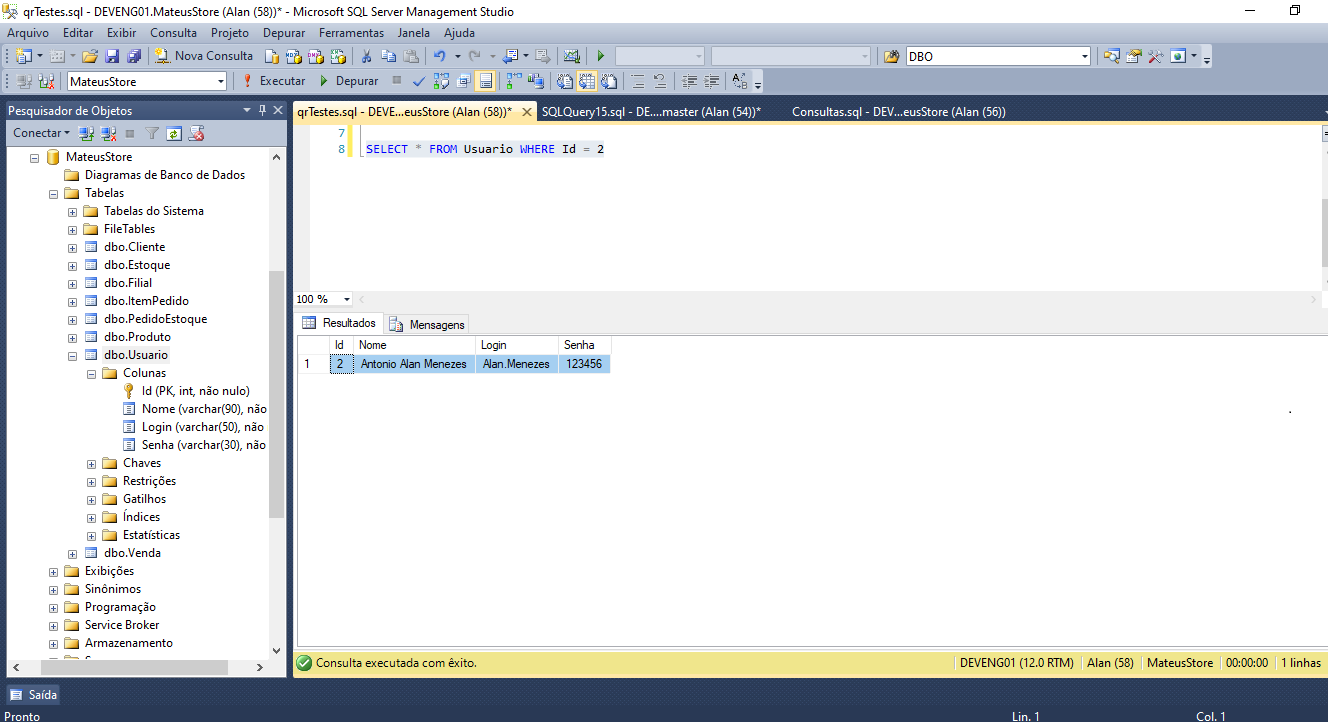
Resultado da requisição

Verificando registro na base de dados

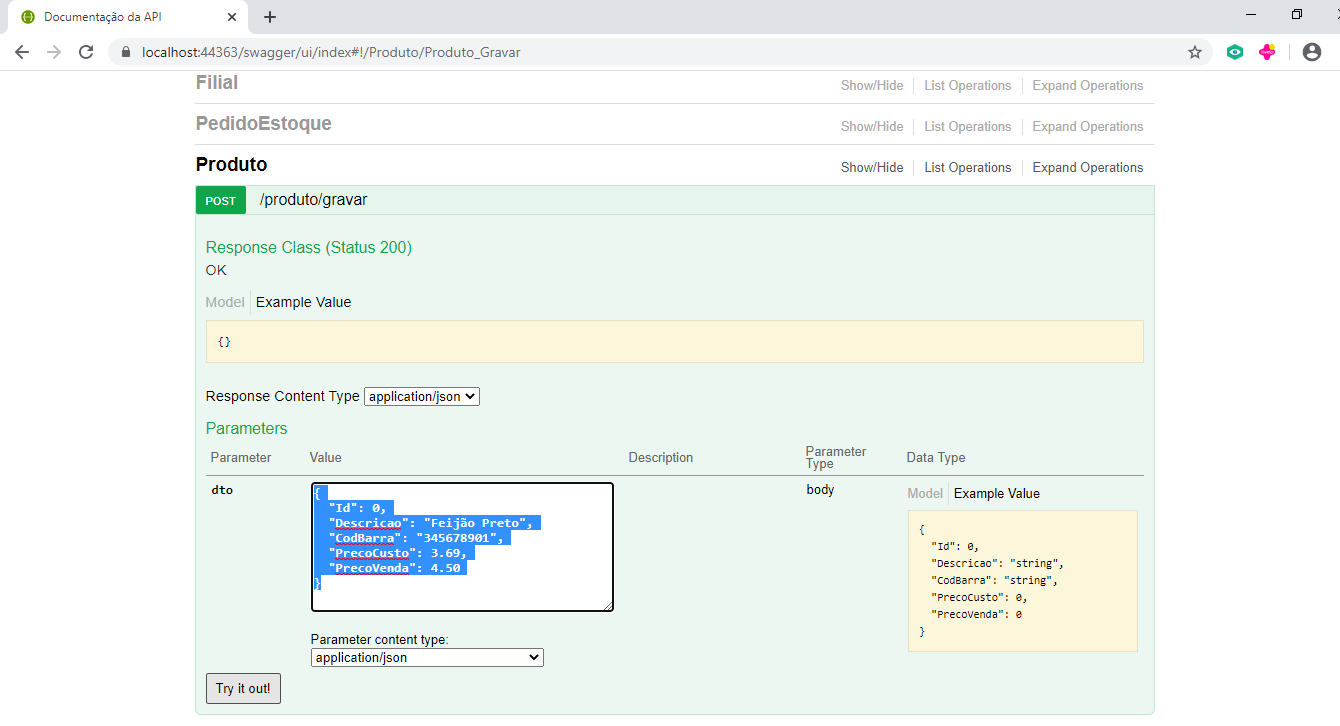
### Gravando e Buscando Usuário

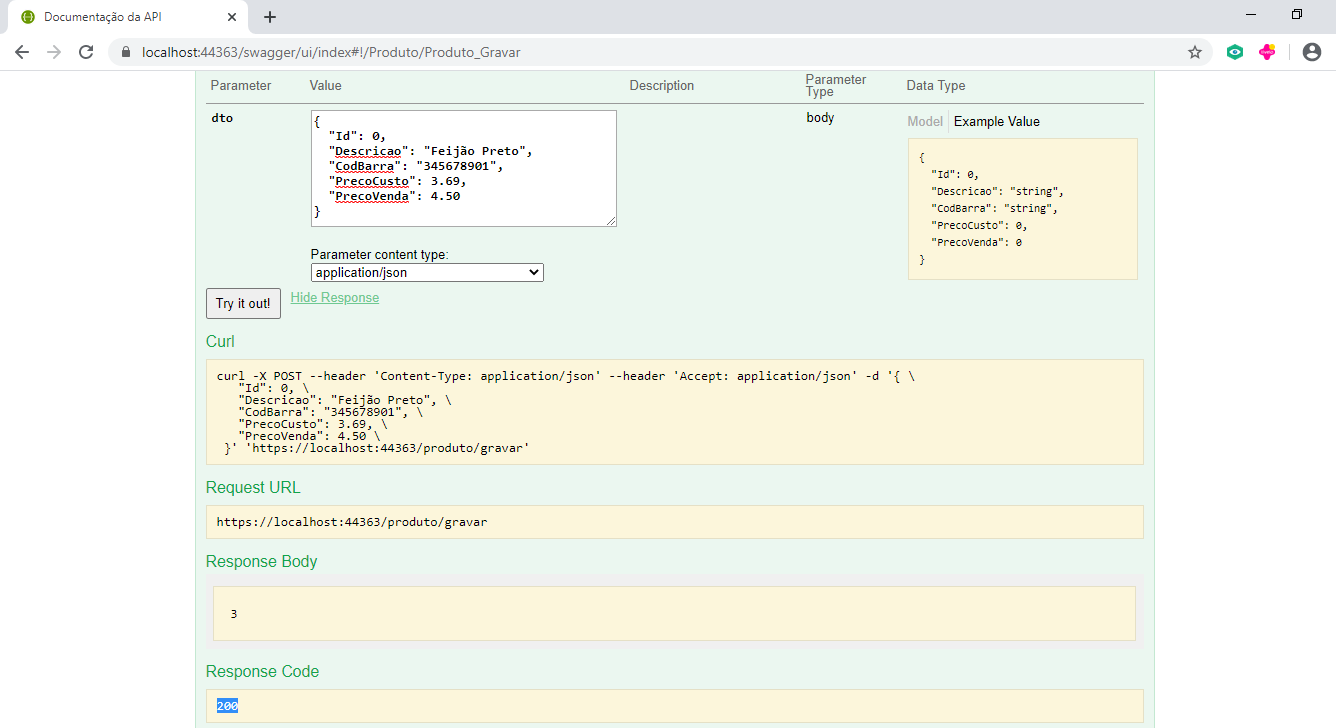


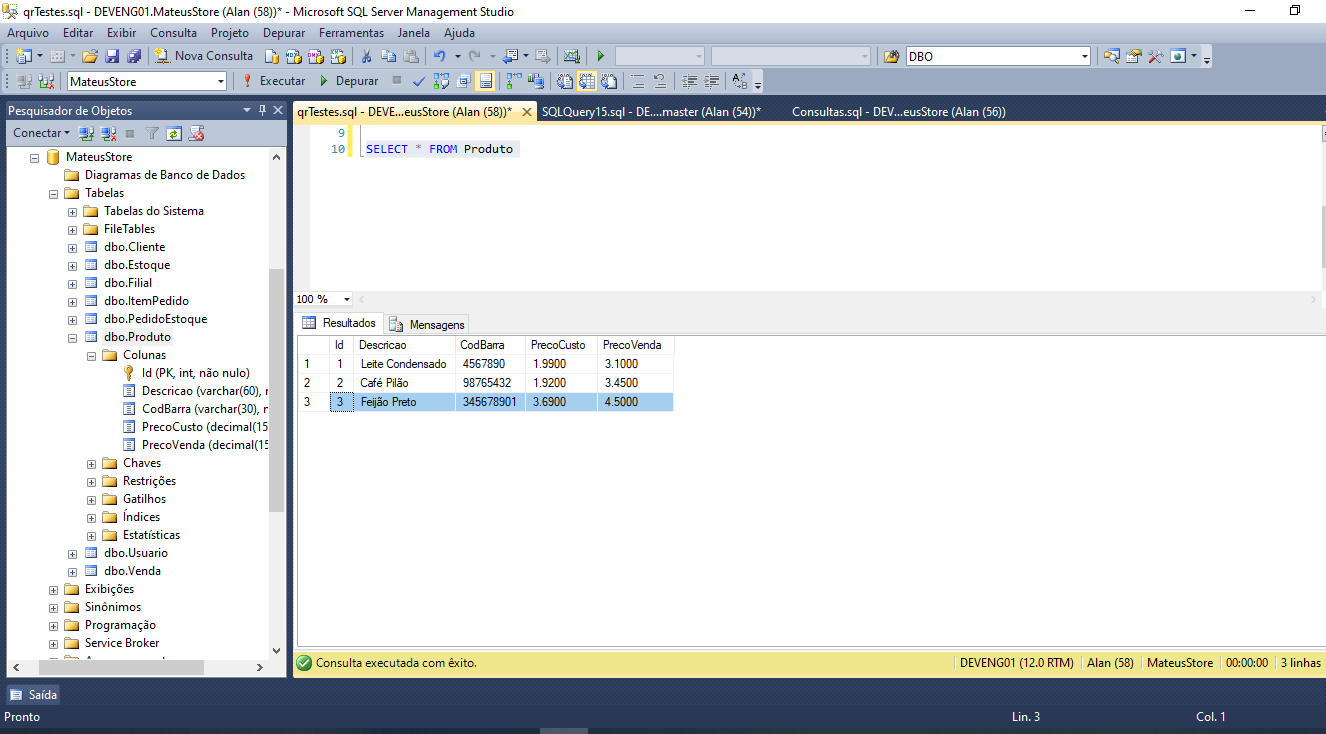
Resultado da requisição

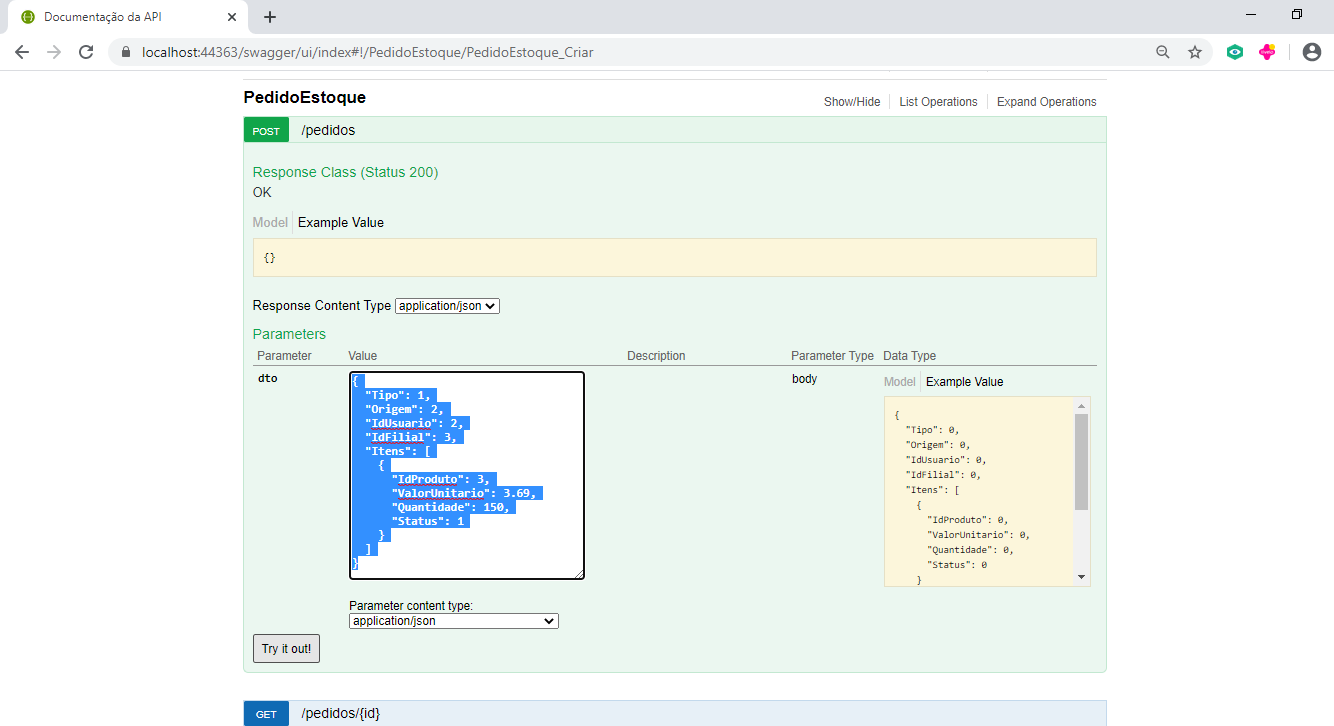
Verificando registro na base de dados

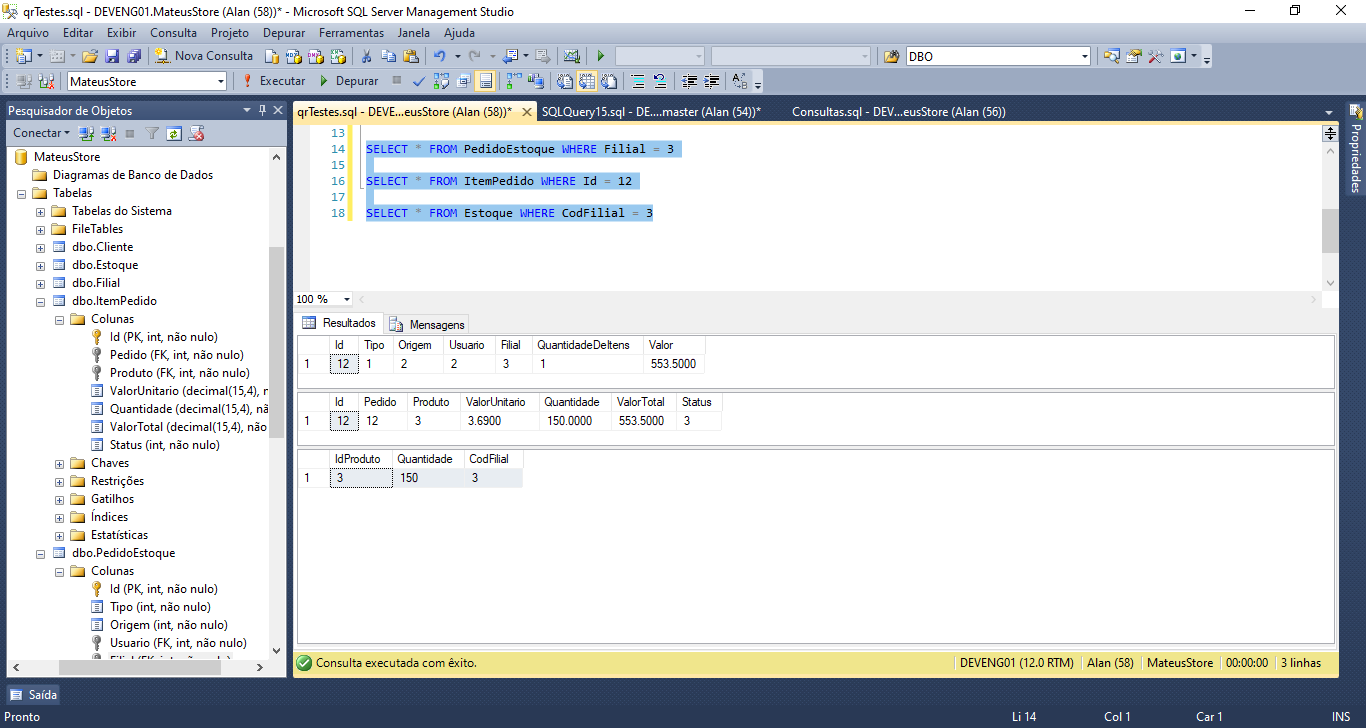
### Gravando e Buscando Produto



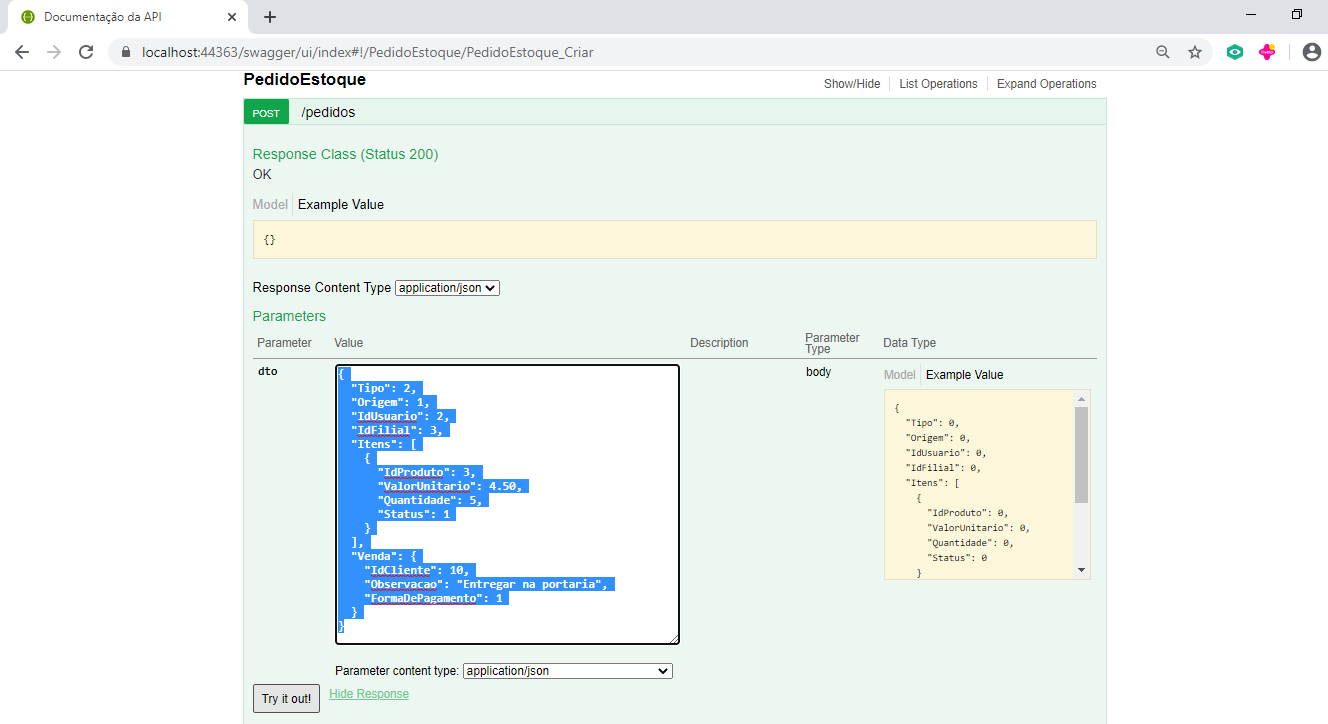
Resultado da requisição

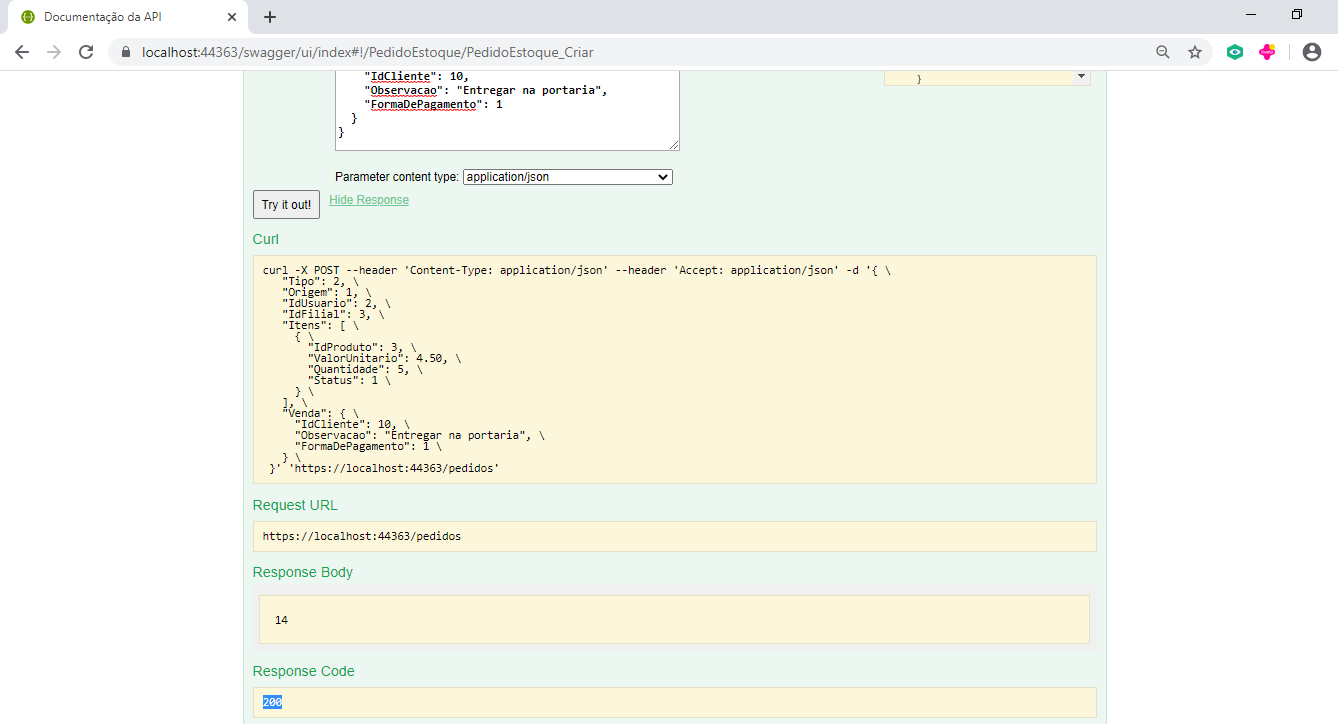
Verificando registro na base de dados

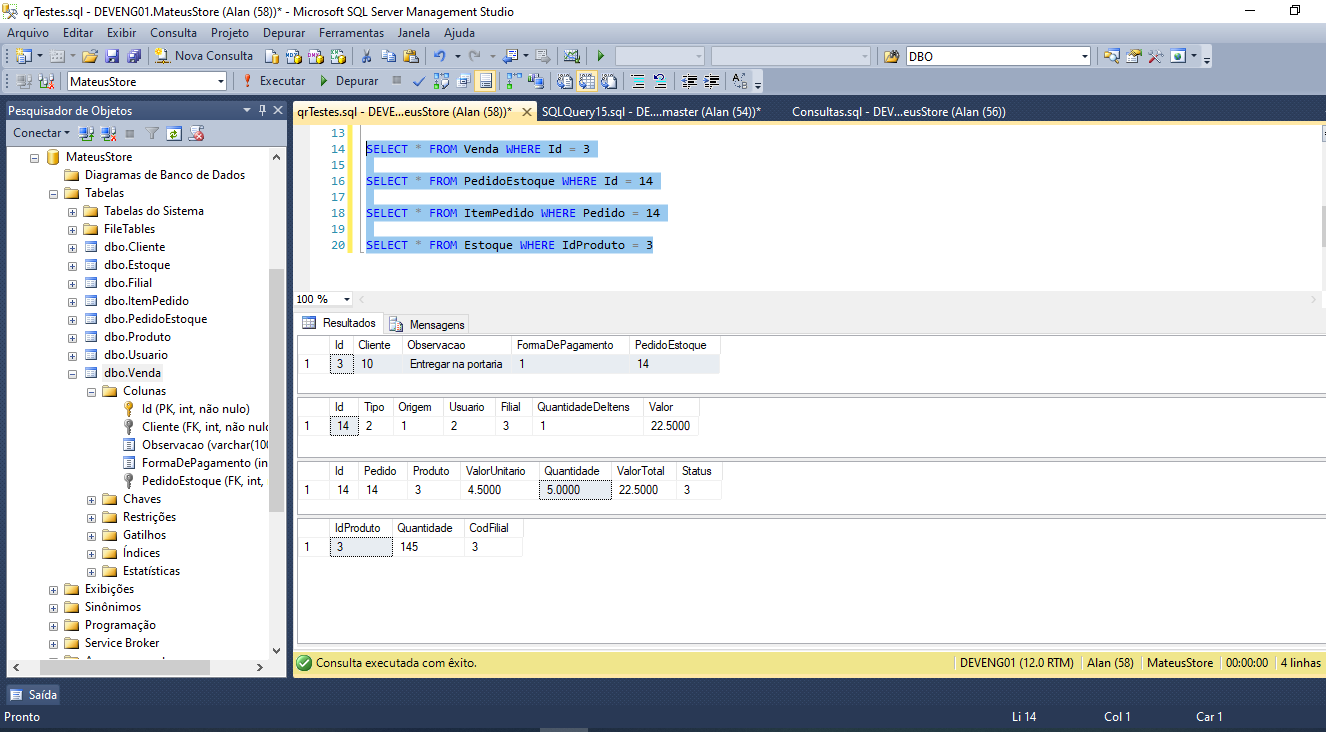
Realizando Pedido de Entrada

  
Verificando registro na base de dados

Realizando Pedido de Saída (Venda)



Resultado da requisição

Verificando registro na base de dados