**Especificação Técnica – Microserviços – Desafio Compasso**

*05/07/2021*

**Índice**

[1. INTRODUÇÃO 4](#_Toc76407097)

[1.1 Objetivos e Definições 4](#_Toc76407098)

[1.2 Público Alvo 4](#_Toc76407099)

[2 Microserviços – Desafio Compasso 5](#_Toc76407100)

[2.1 Métodos 5](#_Toc76407101)

[2.1.1 saveProduto – [POST] 5](#_Toc76407102)

[2.1.1.1 Descrição 5](#_Toc76407103)

[2.1.1.2 Assinatura do Método 5](#_Toc76407104)

[2.1.1.3 Parâmetros de Entrada 5](#_Toc76407105)

[2.1.1.4 Retorno do Método 5](#_Toc76407106)

[2.1.1.5 Regras de Negócios 6](#_Toc76407107)

[2.1.1.6 Execução do teste JUnit 7](#_Toc76407108)

[2.1.1.7 Execução do teste manual 8](#_Toc76407109)

[2.1.2 upDateProduto – [PUT] 8](#_Toc76407110)

[2.1.2.1 Descrição 8](#_Toc76407111)

[2.1.2.2 Assinatura do Método 9](#_Toc76407112)

[2.1.2.3 Parâmetros de Entrada 9](#_Toc76407113)

[2.1.2.4 Retorno do Método 9](#_Toc76407114)

[2.1.2.5 Regras de Negócios 10](#_Toc76407115)

[2.1.2.6 Execução do teste JUnit 11](#_Toc76407116)

[2.1.2.7 Execução do teste manual 12](#_Toc76407117)

[2.1.3 getProdutoById – [GET] 15](#_Toc76407118)

[2.1.3.1 Descrição 15](#_Toc76407119)

[2.1.3.2 Assinatura do Método 15](#_Toc76407120)

[2.1.3.3 Parâmetros de Entrada 15](#_Toc76407121)

[2.1.3.4 Retorno do Método 15](#_Toc76407122)

[2.1.3.5 Regras de Negócios 16](#_Toc76407123)

[2.1.3.6 Execução do teste JUnit 17](#_Toc76407124)

[2.1.3.7 Execução do teste manual 18](#_Toc76407125)

[2.1.4 getAllProdutos – [GET] 19](#_Toc76407126)

[2.1.4.1 Descrição 19](#_Toc76407127)

[2.1.4.2 Assinatura do Método 19](#_Toc76407128)

[2.1.4.3 Parâmetros de Entrada 19](#_Toc76407129)

[2.1.4.4 Retorno do Método 20](#_Toc76407130)

[2.1.4.5 Regras de Negócios 21](#_Toc76407131)

[2.1.4.6 Execução do teste JUnit 21](#_Toc76407132)

[2.1.4.7 Execução do teste manual 22](#_Toc76407133)

[2.1.5 getProdutoSearch – [GET] 23](#_Toc76407134)

[2.1.5.1 Descrição 23](#_Toc76407135)

[2.1.5.2 Assinatura do Método 23](#_Toc76407136)

[2.1.5.3 Parâmetros de Entrada 23](#_Toc76407137)

[2.1.5.4 Retorno do Método 24](#_Toc76407138)

[2.1.5.5 Regras de Negócios 25](#_Toc76407139)

[2.1.5.6 Execução do teste JUnit 25](#_Toc76407140)

[2.1.5.7 Execução do teste manual 27](#_Toc76407141)

[2.1.6 deleteProduto – [DELETE] 35](#_Toc76407142)

[2.1.6.1 Descrição 35](#_Toc76407143)

[2.1.6.2 Assinatura do Método 35](#_Toc76407144)

[2.1.6.3 Parâmetros de Entrada 35](#_Toc76407145)

[2.1.6.4 Retorno do Método 35](#_Toc76407146)

[2.1.6.5 Regras de Negócios 36](#_Toc76407147)

[2.1.6.6 Execução do teste JUnit 36](#_Toc76407148)

[2.1.6.7 Execução do teste manual 37](#_Toc76407149)

[2.2 Erros e Mensagens 39](#_Toc76407150)

[2.3 Execução NUnit completa: 39](#_Toc76407151)

# INTRODUÇÃO

## Objetivos e Definições

Este documento tem como objetivo mapear os métodos disponibilizados no microserviço do desafio da compasso e disponibilizar as requisições que serão necessários para utilização do serviço com detalhe das requisições.

## Público Alvo

Todos os canais externos ou internos que utilizam os microserviços informados nesta documentação.

# Microserviços – Desafio Compasso

## Métodos

### saveProduto – [POST]

#### Descrição

Esse endpoint é responsável por criar um novo produto na base de dados, ao receber como entrada o JSON com os campos e dados do produto.

Caso o JSON esteja válido, o mesmo é persistido na base de dados e nos é retornado o status com o código HTTP 201.

* **Link localhost:** 
  + [**http://localhost:9999/products**](http://localhost:9999/products)

#### Assinatura do Método

public ResponseEntity<DesafioCompasso> saveProduto(@RequestBody DesafioCompasso produto);

#### Parâmetros de Entrada

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | produto | DesafioCompasso | Objeto responsável por conter os campos do produto. |

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| ResponseEntity<DesafioCompasso> | Response do objeto DesafioCompasso. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

**JSON – Entrada:**

|  |
| --- |
| {  "name": "NAME 001",  "description": "DESCRITION 001",  "price": 1.0  } |

**JSON – Saída:**

|  |
| --- |
| {  "id": 1,  "name": "NAME 001",  "description": "DESCRITION 001",  "price": 1  } |

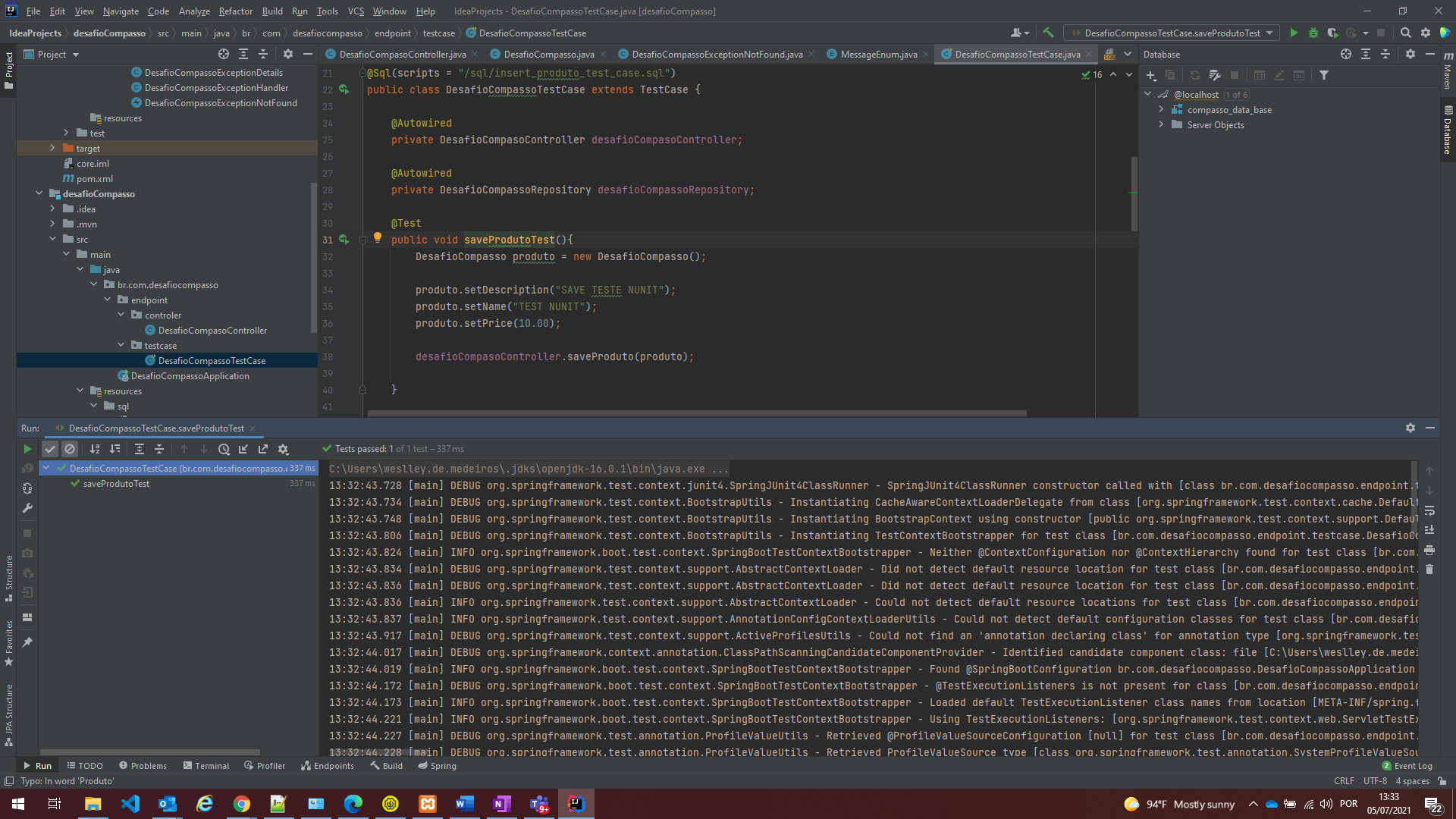
**JSON – Status 400:**

|  |
| --- |
| {  "status\_code": 400,  "message": "Ocorreu um erro ao gravar os dados, verifique os campos informados e tente novamente."  } |

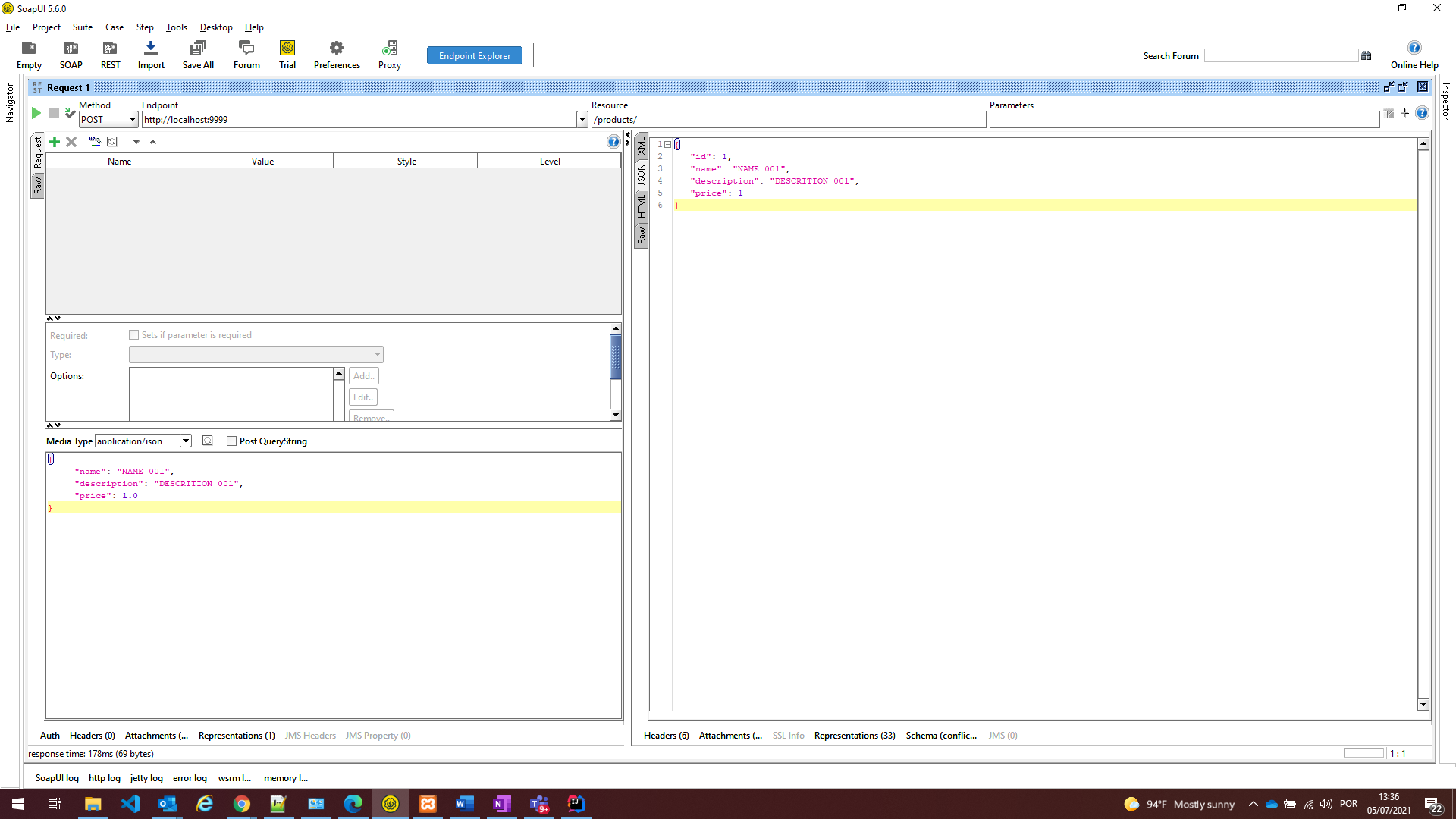
#### Regras de Negócios

* Os campos name, description e price são obrigatórios.
* O campo price deve possuir o seu valor sempre positivo.
* Em caso de algum erro de validação, a API deve retornar um HTTP 400 indicando uma requisição inválida.
* No caso de status 400 o campo status\_code deve vir o código HTTP do erro de validação (400 Bad Request) e no campo message deverá vir uma mensagem genérica indicando o erro ocorrido.

#### Execução do teste JUnit



#### Execução do teste manual



### upDateProduto – [PUT]

#### Descrição

Esse endpoint é responsável por atualizar um produto baseado no seu {id} passado via path param.

Antes de alterar o sistema realiza uma consultada a base de dados pelo id, se o produto NÃO for localizado o microserviço retorna HTTP 404 ao cliente.

Se localizar o produto, então os campos name, description e price devem ser atualizados conforme recebidos no body da requisição.

* **Link localhost:** 
  + [**http://localhost:9999/products/{id}**](http://localhost:9999/products/%7bid%7d)

#### Assinatura do Método

public ResponseEntity<DesafioCompasso> upDateProduto(@PathVariable(value = "id") Long id, @Valid @RequestBody DesafioCompasso produto)

#### Parâmetros de Entrada

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | produto | DesafioCompasso | Objeto responsável por conter os campos do produto. |

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| ResponseEntity<DesafioCompasso> | Response do objeto DesafioCompasso. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

**JSON – Entrada:**

|  |
| --- |
| {  "name": "NAME 001 ALTER",  "description": "DESCRITION 001 ALTER",  "price": 1.89  } |

**JSON – Saída:**

|  |
| --- |
| {  "id": 1,  "name": "NAME 001 ALTER",  "description": "DESCRITION 001 ALTER",  "price": 1.89  } |

**JSON – Status 400:**

|  |
| --- |
| {  "status\_code": 400,  "message": "Ocorreu um erro ao gravar os dados, verifique os campos informados e tente novamente."  } |

**JSON – Status 404:**

|  |
| --- |
| {  "status\_code": 404,  "message": "Produto com ID: 99 inexistente."  } |

#### Regras de Negócios

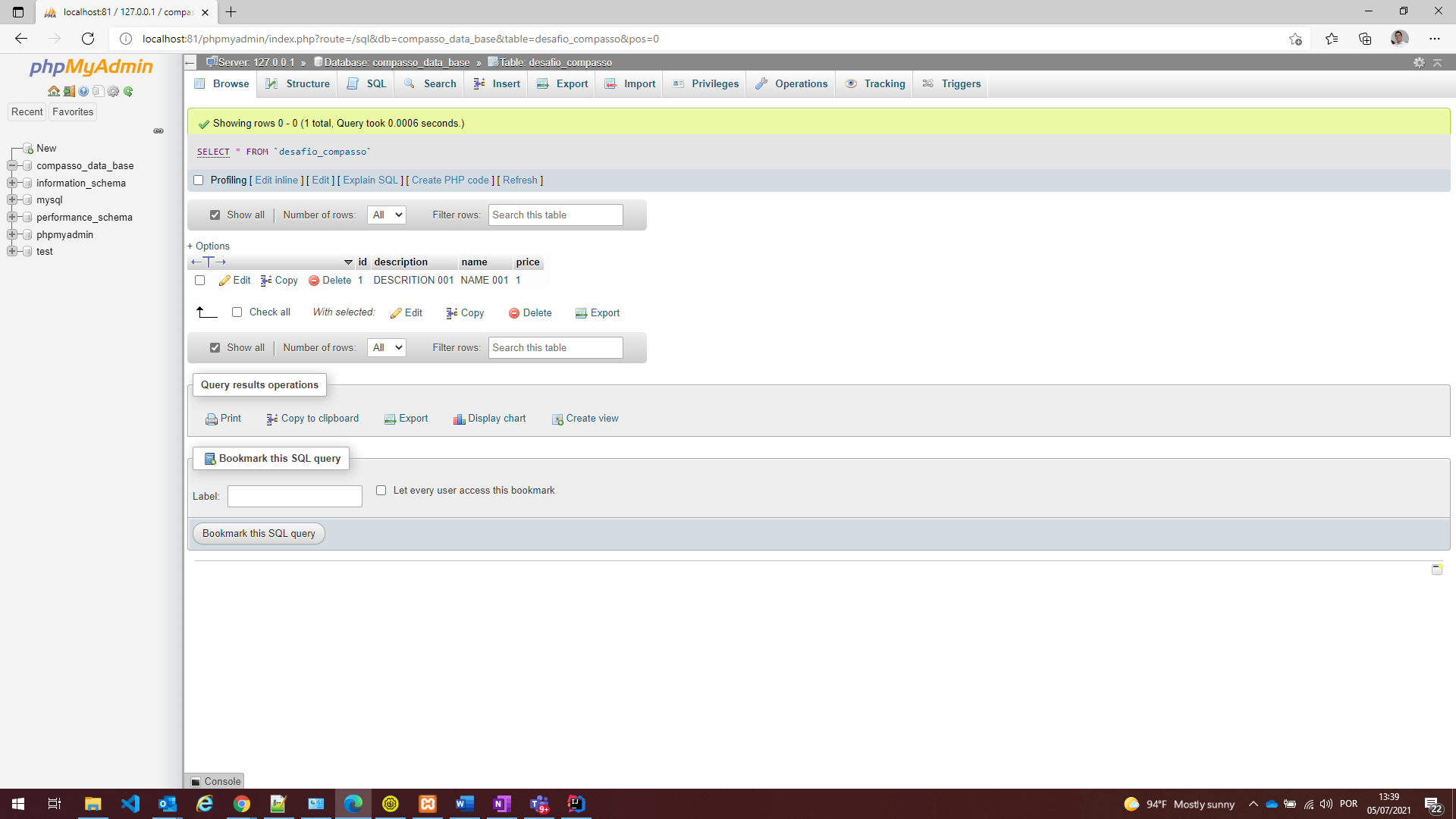
* Os campos name, description e price são obrigatórios.
* O campo price deve possuir o seu valor sempre positivo.
* Antes de alterar o sistema realiza uma consultada a base de dados pelo id, se o produto NÃO for localizado o microserviço retorna HTTP 404 ao cliente.
* Em caso de algum erro de validação, a API deve retornar um HTTP 400 indicando uma requisição inválida.
* No caso de status 400 o campo status\_code deve vir o código HTTP do erro de validação (400 Bad Request) e no campo message deverá vir uma mensagem genérica indicando o erro ocorrido

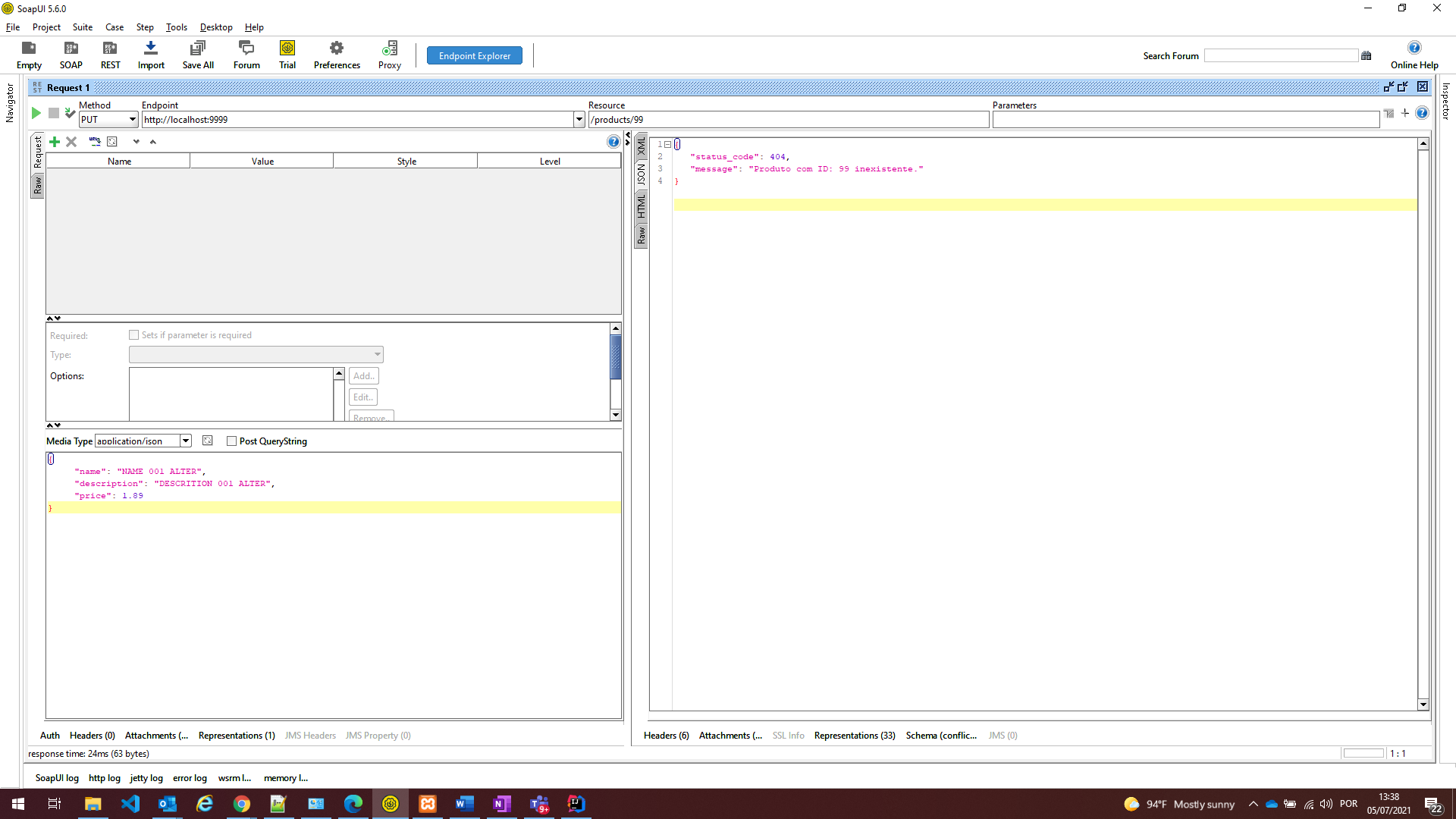
#### Execução do teste JUnit



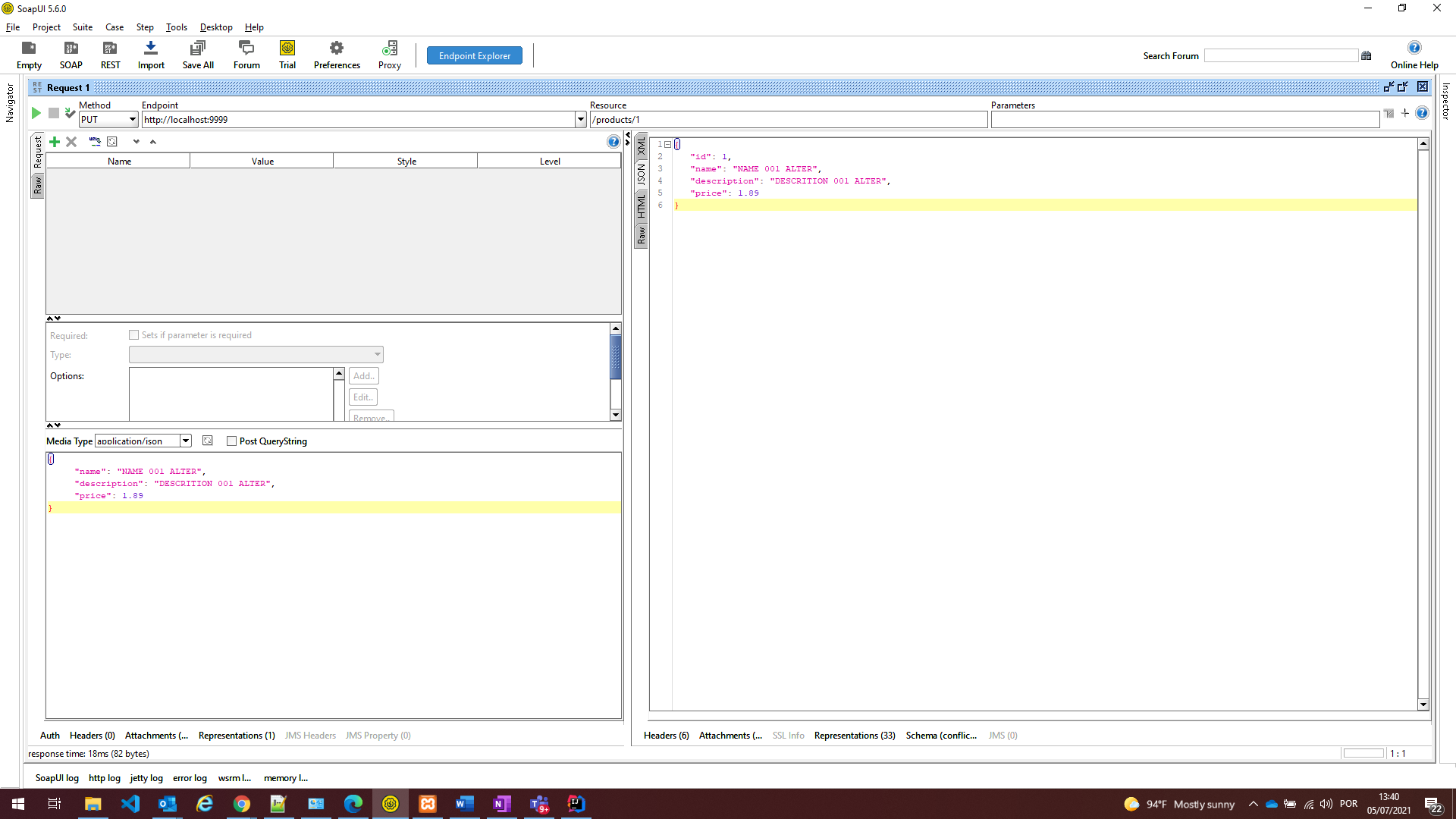
#### Execução do teste manual

* Passando Id inexistente na base de dados:

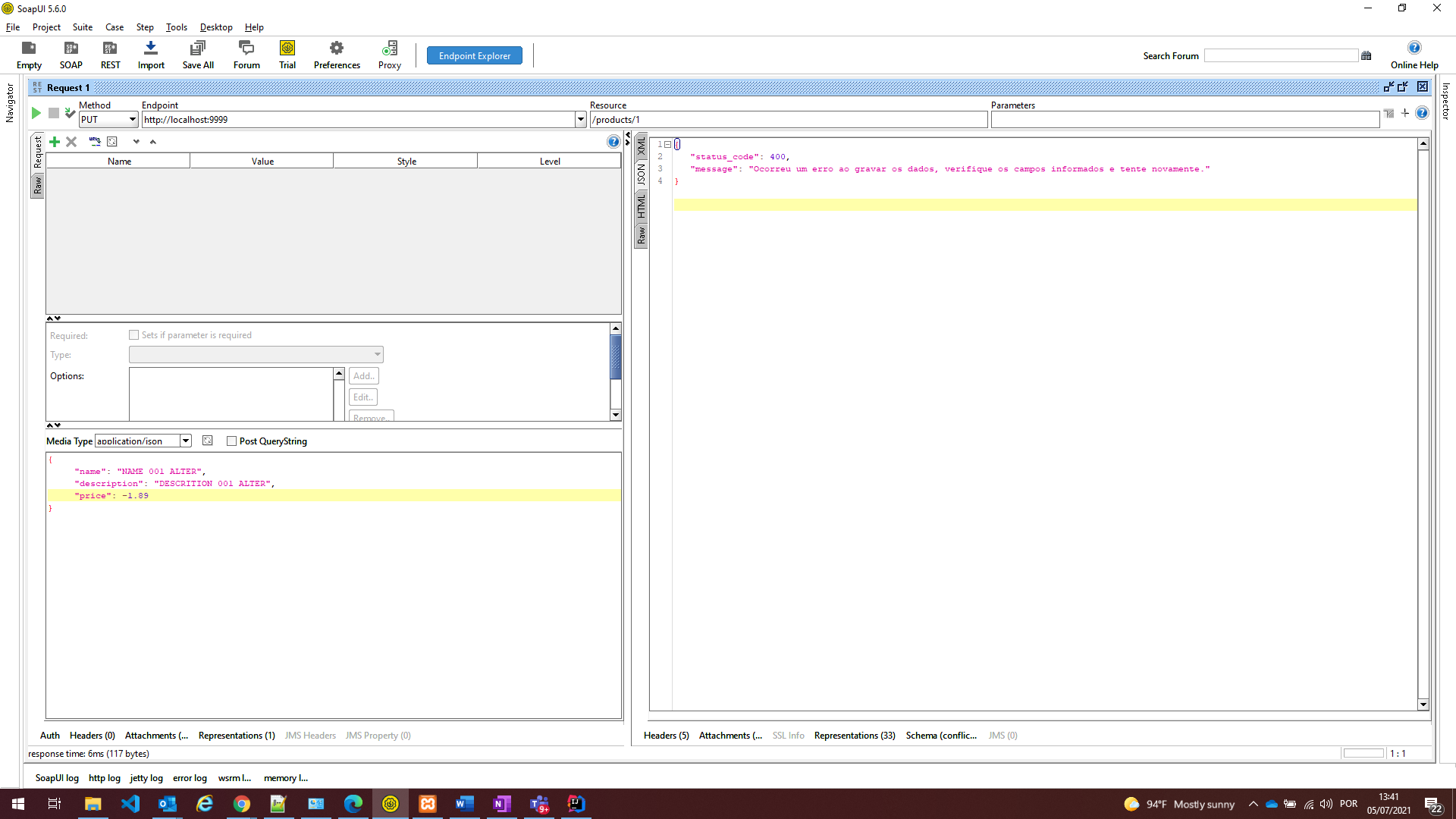




* Passando Id valido:



* Passando “price” negativo:



### getProdutoById – [GET]

#### Descrição

Esse endpoint retorna o produto localizado na base de dados com um HTTP 200, de acordo com o id do produto informado.

Nos casos em que o produto não é localização, a API retorna um HTTP 404 indicando que o produto não foi localizado.

* **Link localhost:** 
  + [**http://localhost:9999/products/{id}**](http://localhost:9999/products/%7bid%7d)

#### Assinatura do Método

public DesafioCompasso getProdutoById(@PathVariable Long id)

#### Parâmetros de Entrada

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| DesafioCompasso | Retorna os dados do produto. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

**JSON – Saída:**

|  |
| --- |
| {  "id": 1,  "name": "NAME 001 ALTER",  "description": "DESCRITION 001 ALTER",  "price": 1.89  } |

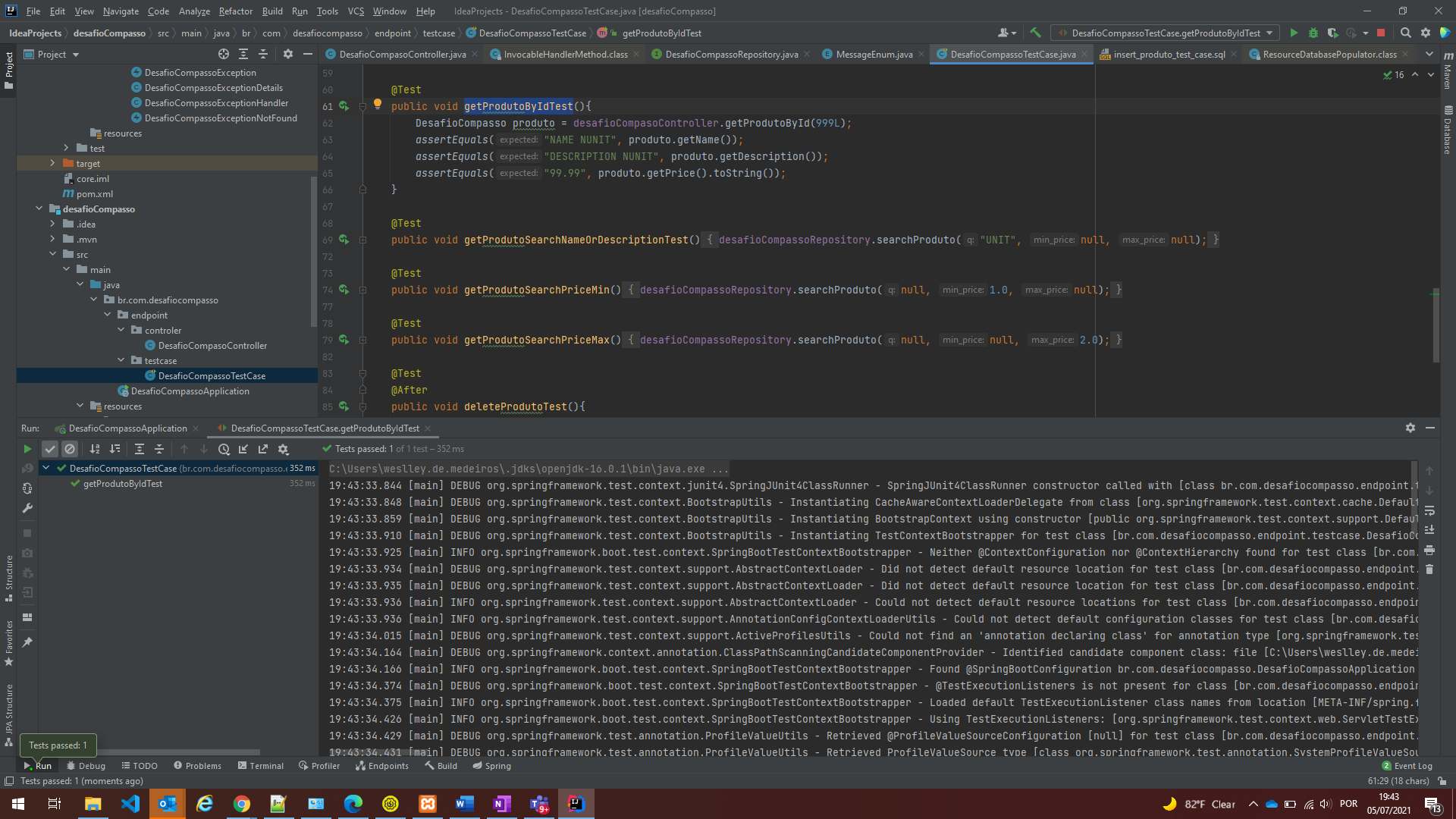
**JSON – Status 404:**

|  |
| --- |
| {  "status\_code": 404,  "message": "Produto com ID: 99 inexistente."  } |

#### Regras de Negócios

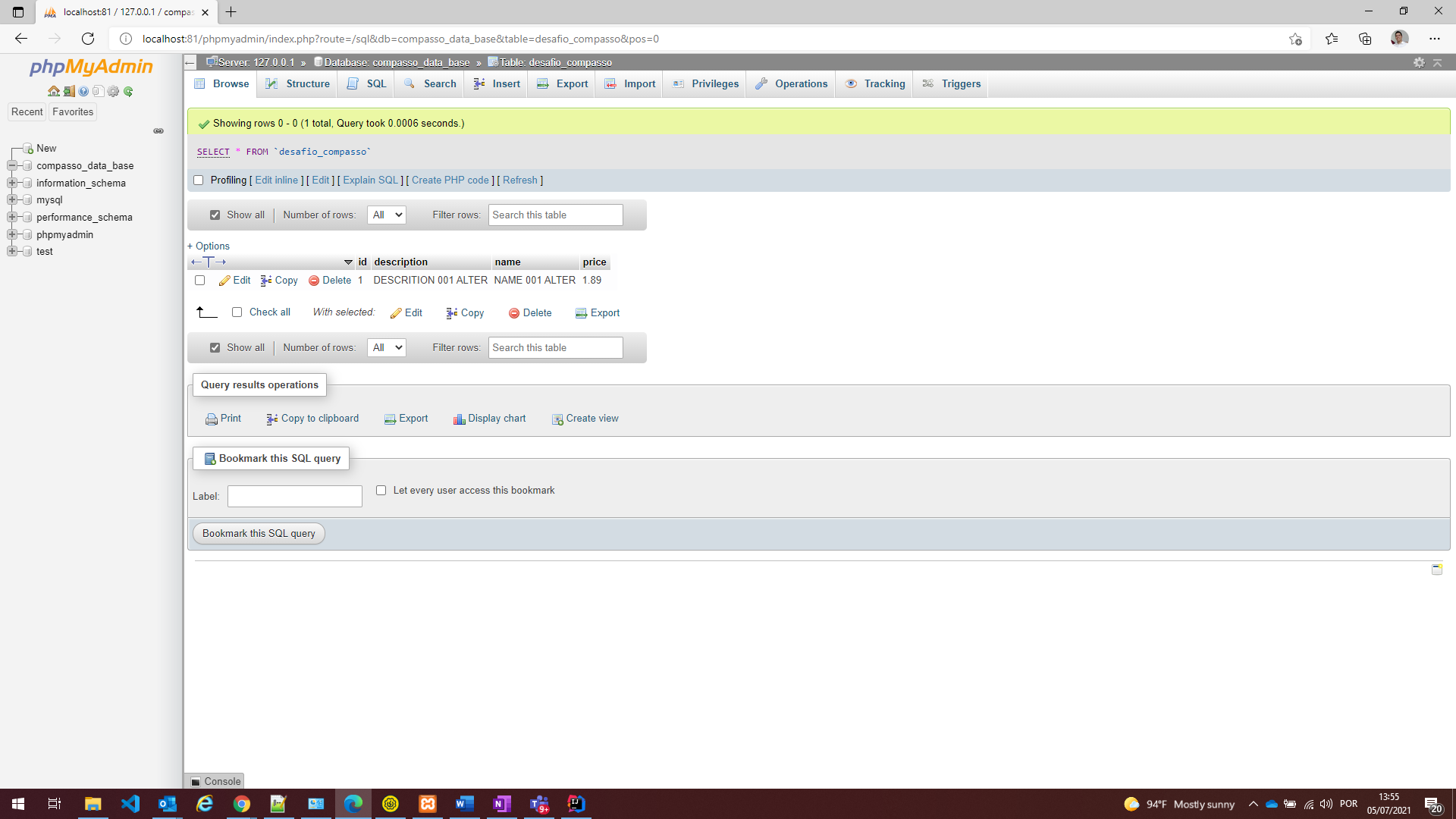
* Consulta realizada a partir do id do produto informado via GET, caso o id exista o JSON com os dados do protudo é retornado, o status HTTP é 200.
* Caso o id informado não exista na base de dados é retornado o JSON conforme informado acima com o HTTP404.

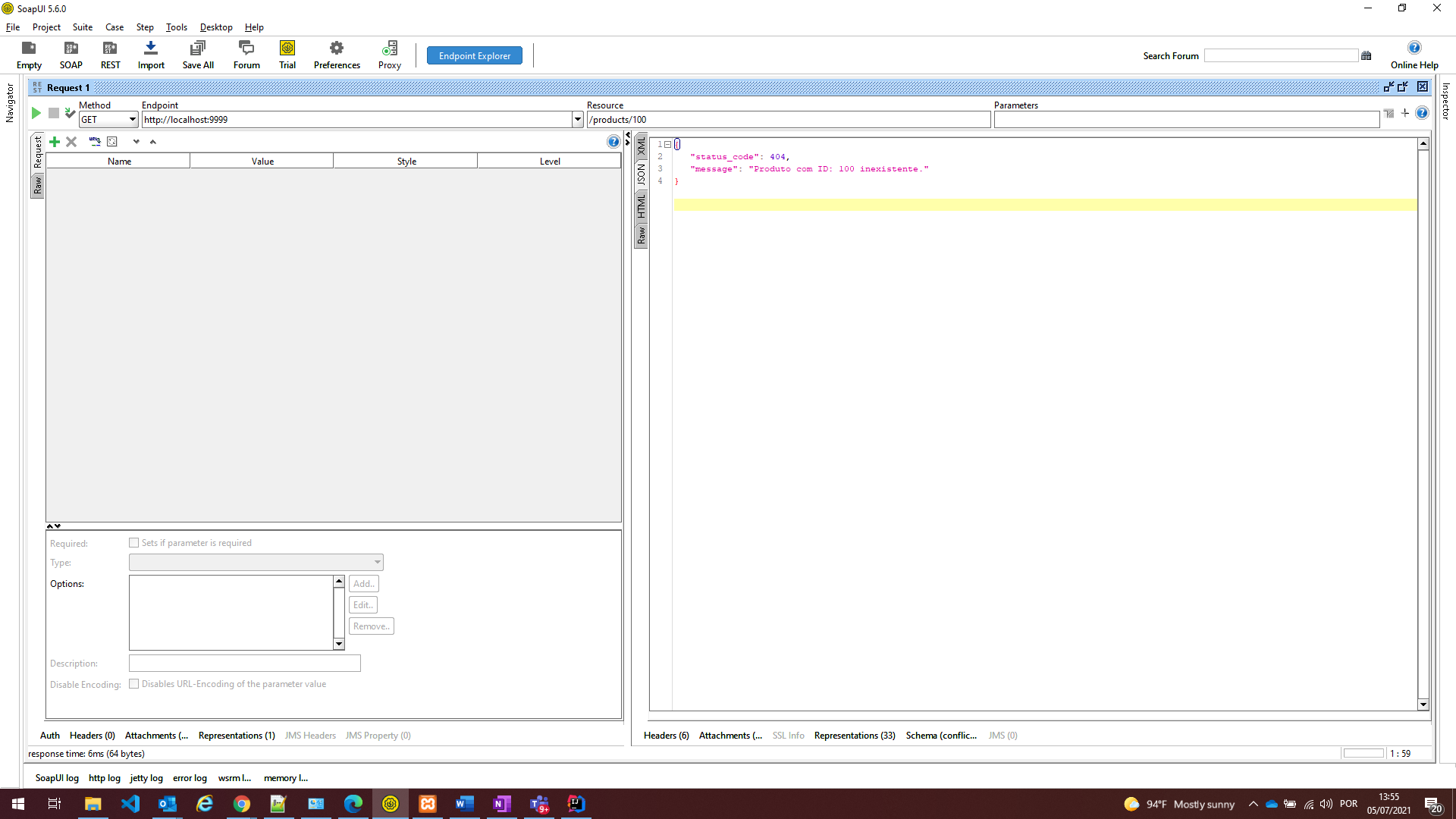
#### Execução do teste JUnit



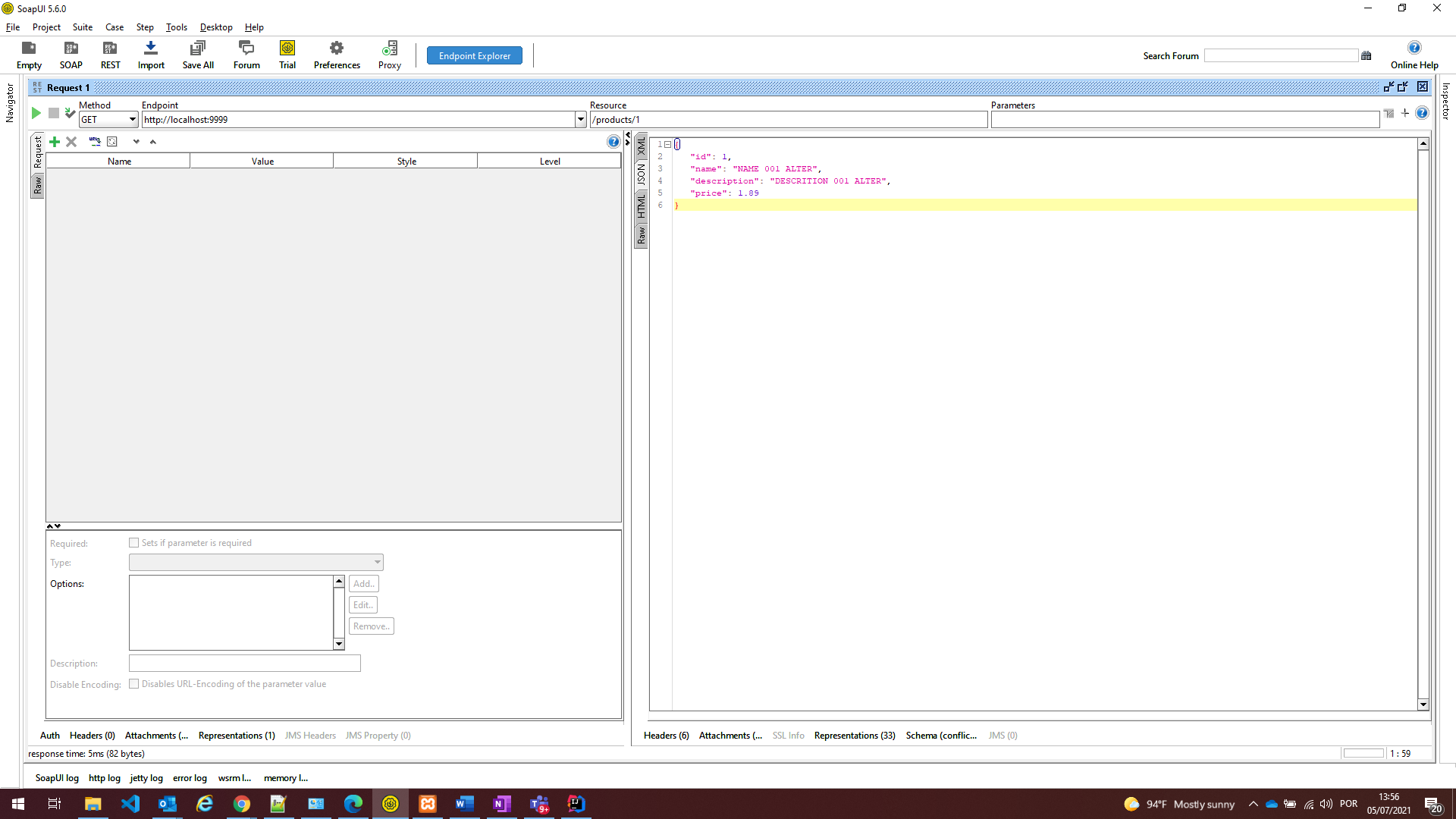
#### Execução do teste manual

* Passando Id inexistente na base de dados:





* Passando Id valido:



### getAllProdutos – [GET]

#### Descrição

Nesse endpoint a API retorna a lista atual de todos os produtos com HTTP 200. Se não existir produtos, retornar uma lista vazia.

* **Link localhost:** 
  + [**http://localhost:9999/products/**](http://localhost:9999/products/)

#### Assinatura do Método

public List<DesafioCompasso> getAllProdutos()

#### Parâmetros de Entrada

N/A – Não se aplica.

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| List<DesafioCompasso> | Lista de todos os produto cadastrados na base. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

**JSON – Saída:**

|  |
| --- |
| [  {  "id": 1,  "name": "NAME 001 ALTER",  "description": "DESCRITION 001 ALTER",  "price": 1.89  },  {  "id": 2,  "name": "NAME 002",  "description": "DESCRITION 002",  "price": 2.89  },  {  "id": 3,  "name": "NAME 003",  "description": "DESCRITION 003",  "price": 3.99  }  ] |

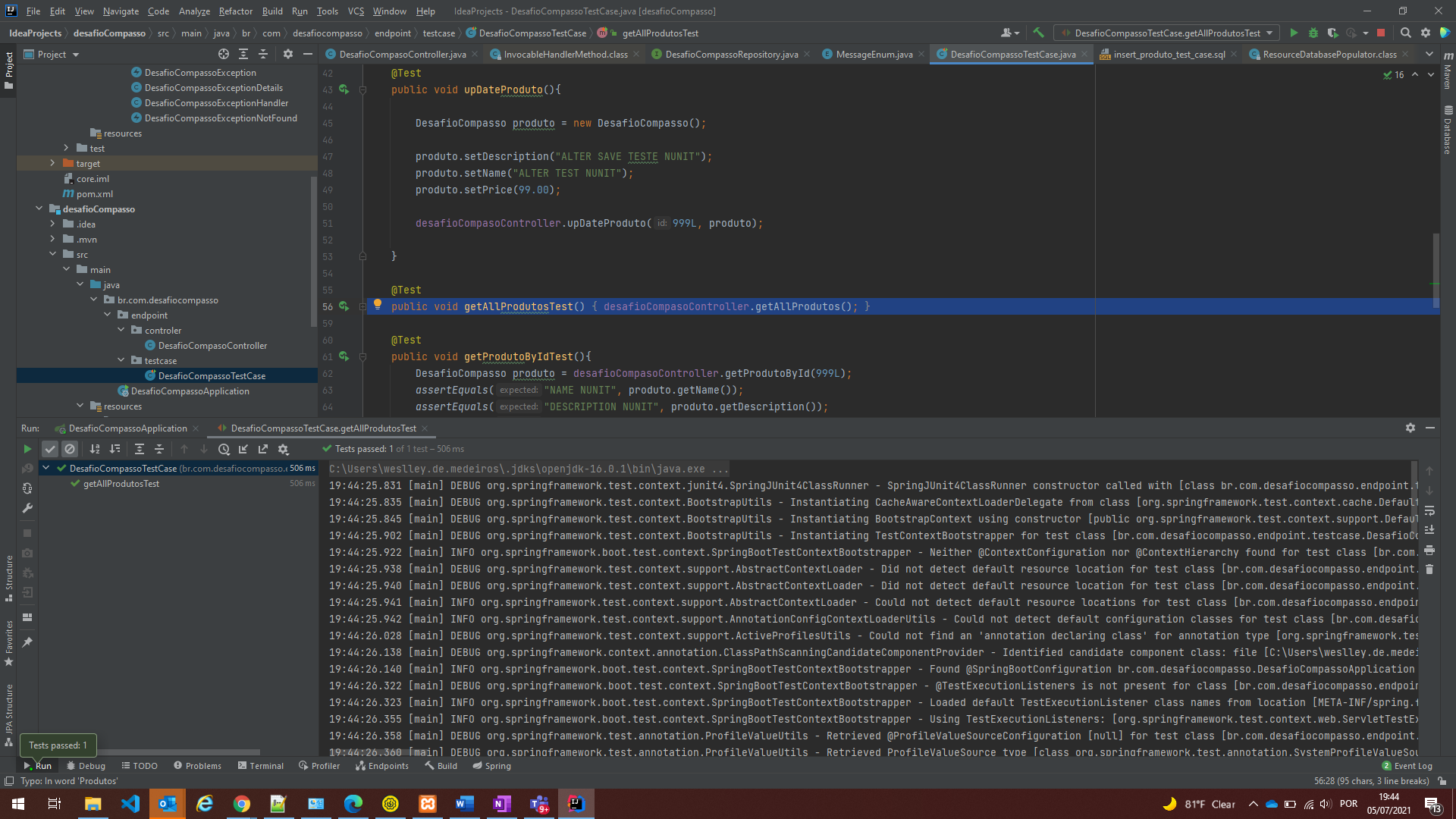
**JSON – Saída, quando não houver produtos retornados:**

|  |
| --- |
| [] |

#### Regras de Negócios

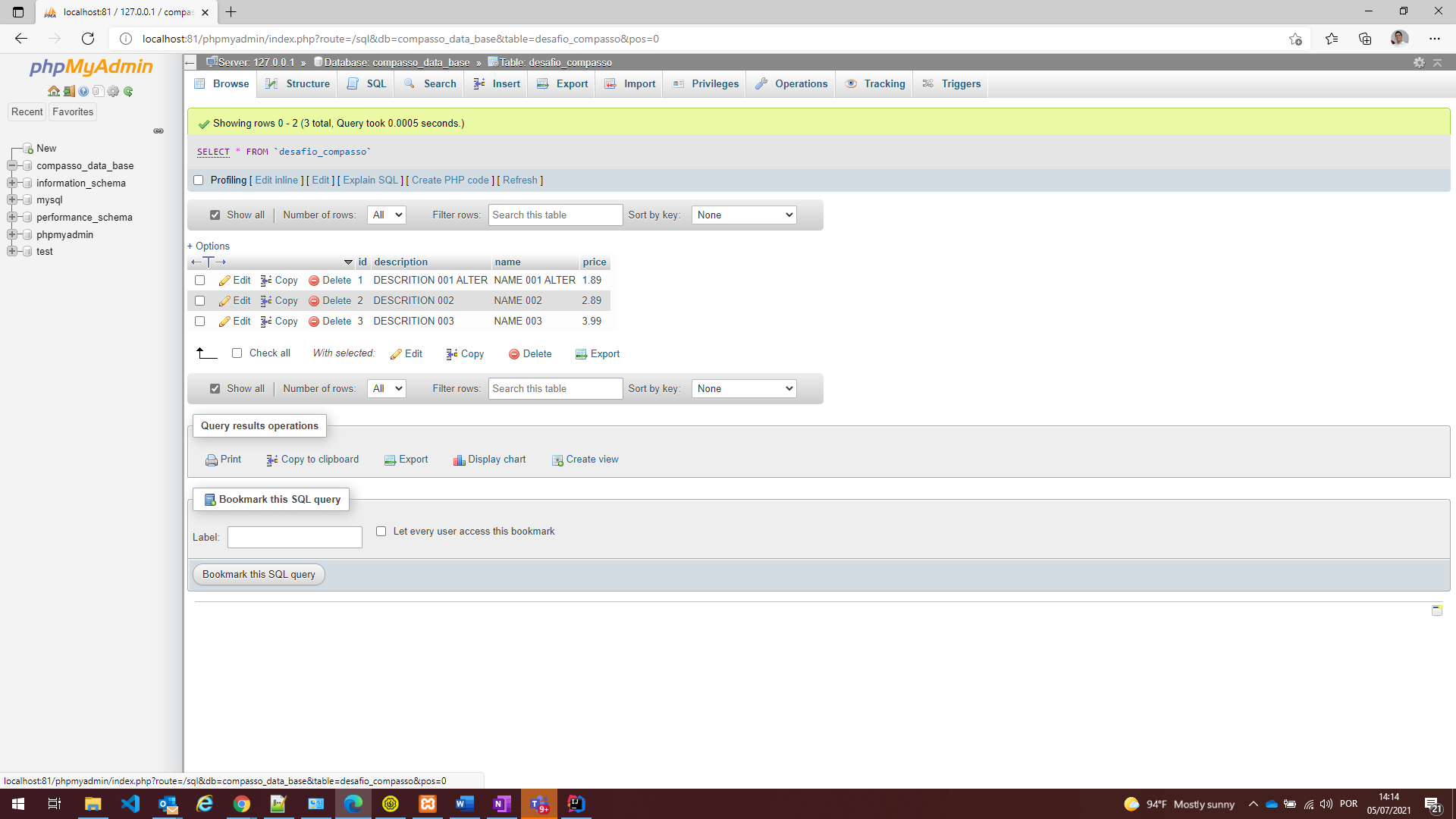
* Retorna todos os produtos listados na base de dados, com o retorno HTTP 200.
* Se não existir produtos, retornar uma lista vazia.

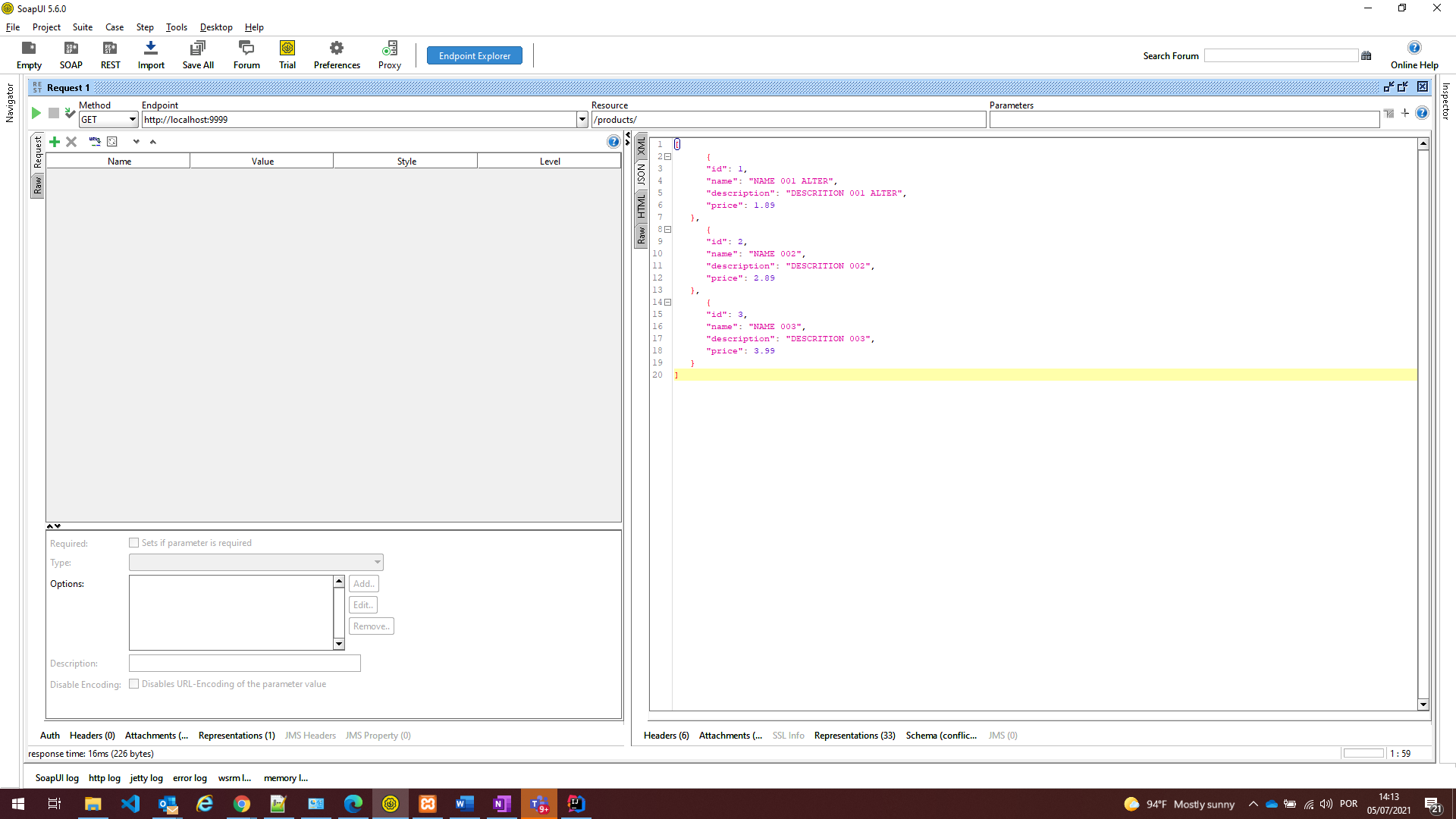
#### Execução do teste JUnit



#### Execução do teste manual

* Lista de todos os produtos cadastrados na base de dados.





### getProdutoSearch – [GET]

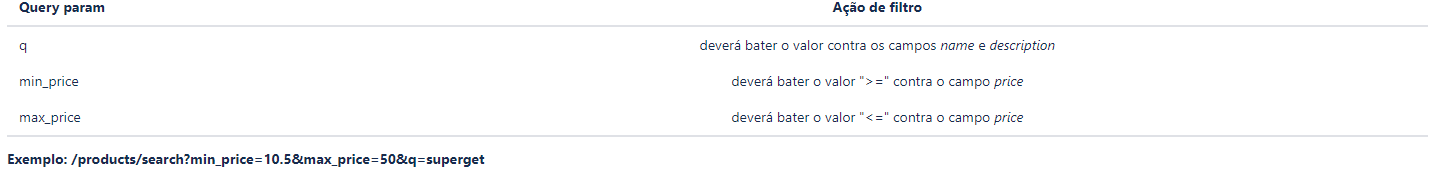
#### Descrição

Nesse endpoint a API retornará a lista de todos os produtos filtrados de acordo com query parameters enviadas via URL.

Os query parameters aceitos serão: q, min\_price e max\_price.

**Importante:** nenhum deles deverá ser obrigatório na requisição

Onde:



* **Link localhost:** 
  + [**http://localhost:9999/products/search**](http://localhost:9999/products/search)

#### Assinatura do Método

public List<DesafioCompasso> getProdutoSearch(@Nullable @RequestParam("q") String q, @Nullable @RequestParam("min\_price") Double min\_price, @Nullable @RequestParam("max\_price") Double max\_price)

#### Parâmetros de Entrada

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | q | String | Correspondem aos campos name e description. |
| 2 | min\_price | Double | Valor mínimo do campo price. |
| 3 | max\_price | Double | Valor máximo do campo price. |

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| List<DesafioCompasso> | Lista de todos os produto cadastrados na base de acordo com o filtro da consulta. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

**JSON – Saída:**

|  |
| --- |
| [  {  "id": 1,  "name": "NAME 001 ALTER",  "description": "DESCRITION 001 ALTER",  "price": 1.89  },  {  "id": 2,  "name": "NAME 002",  "description": "DESCRITION 002",  "price": 2.89  },  {  "id": 3,  "name": "NAME 003",  "description": "DESCRITION 003",  "price": 3.99  }  ] |

**JSON – Saída, quando não houver produtos retornados:**

|  |
| --- |
| [] |

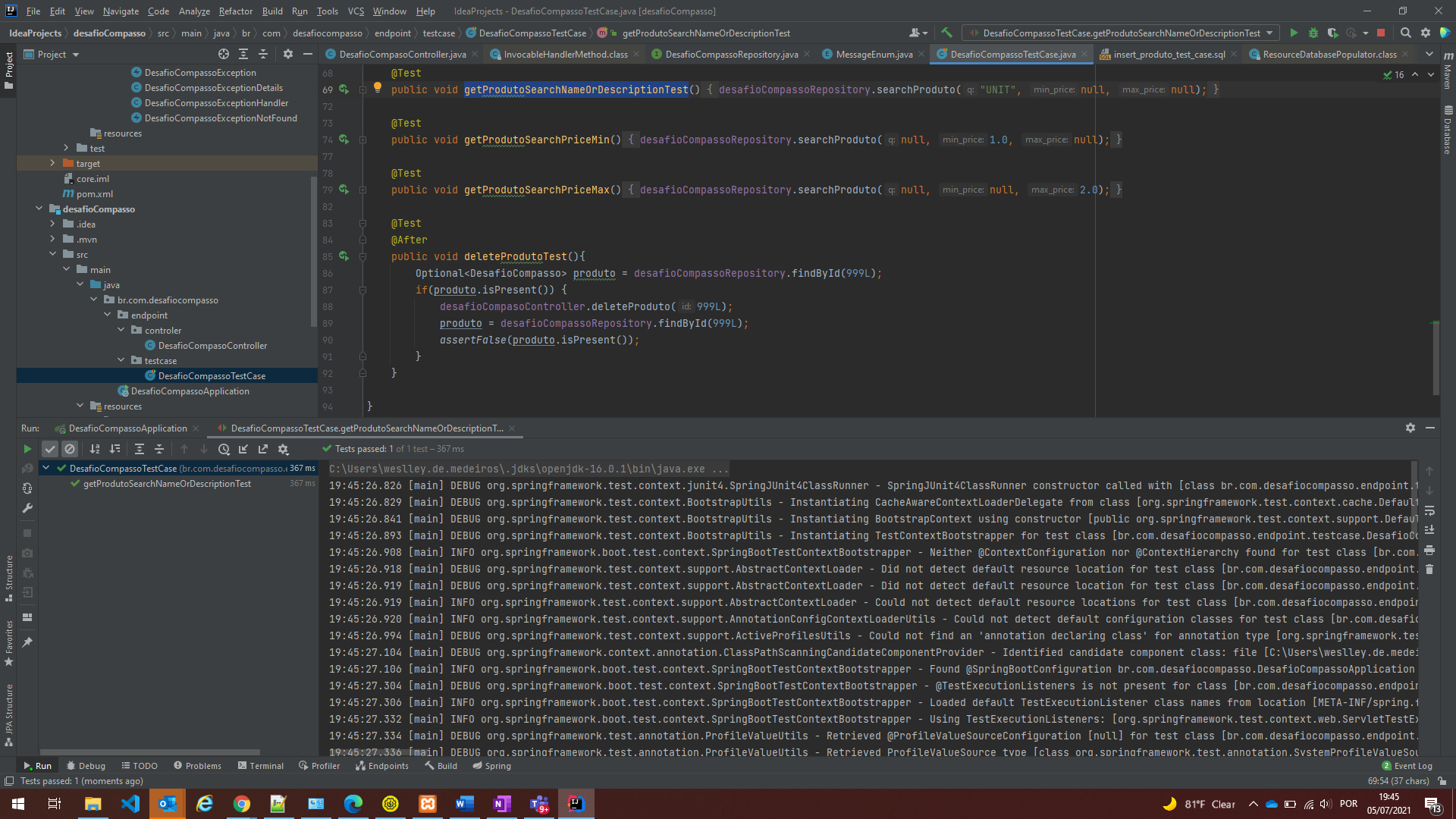
**JSON – Status 400:**

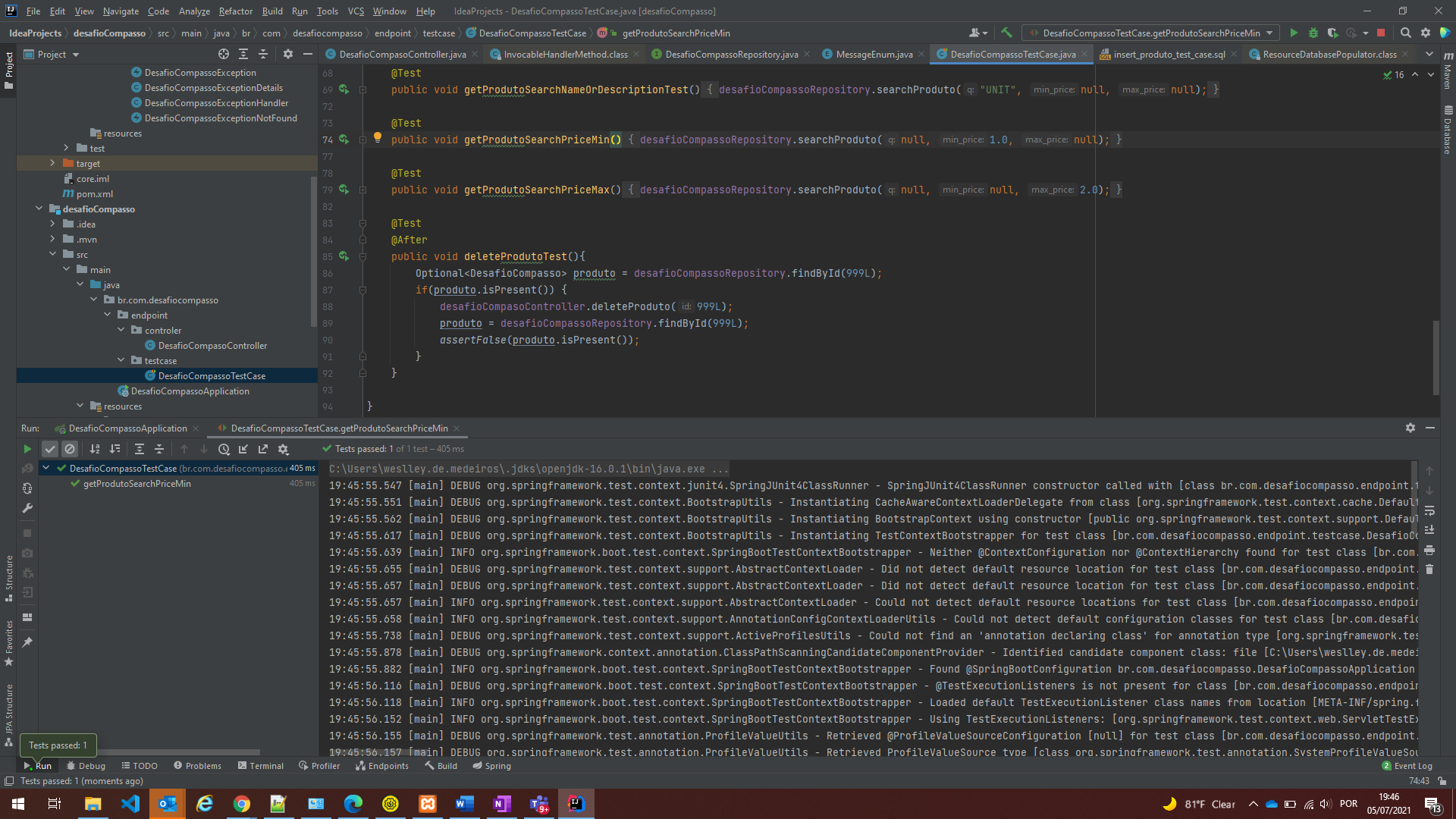
|  |
| --- |
| {  "status\_code": 400,  "message": "Ocorreu um erro nos dados, verifique os campos informados e tente novamente."  } |

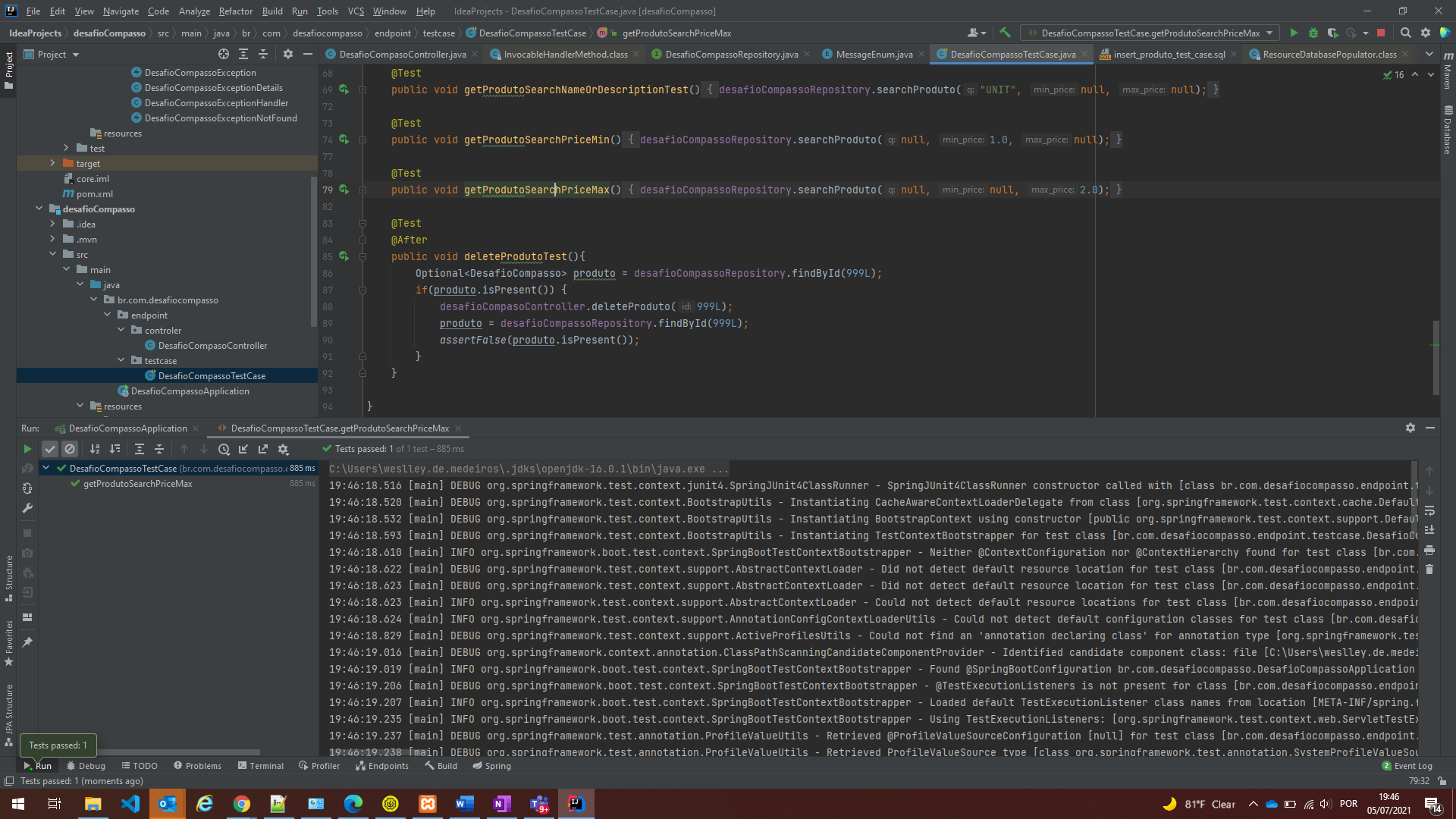
#### Regras de Negócios

* Retorna todos os produtos listados na base de dados, de acordo com o query parameters enviadas via URL.
* Se não existir produtos, retornar uma lista vazia.
* Nos casos em que forem informados os preços minino e máximo é realizada uma validação para que o sistema não permita que o “min\_price” seja maior que o “max\_price”.
* Quando for passado apenas os parâmetros “min\_price” e “max\_price” a query a ser executada será para consultar o intervalo entre os dois valores (BETWEEN).

#### Execução do teste JUnit

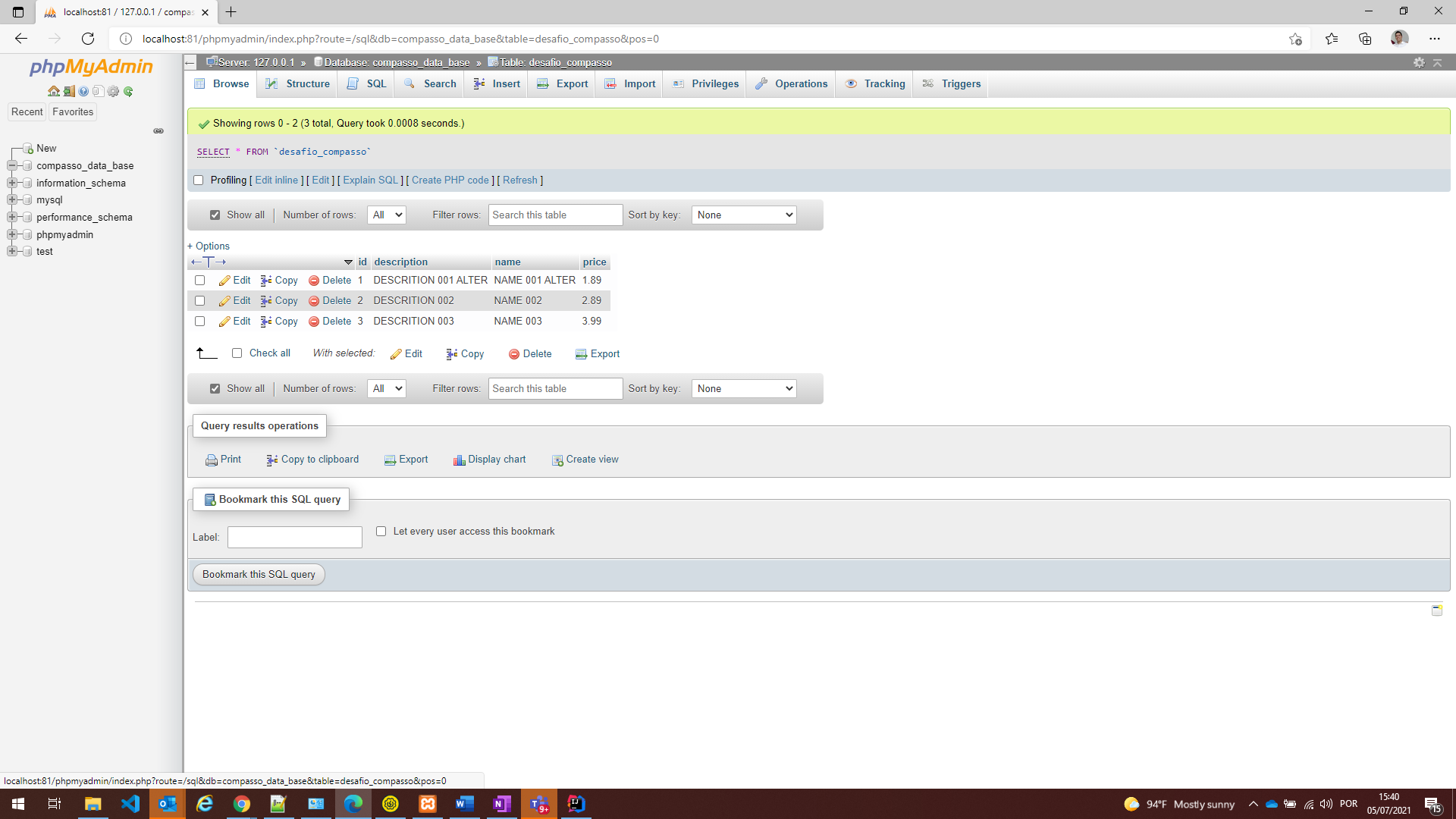




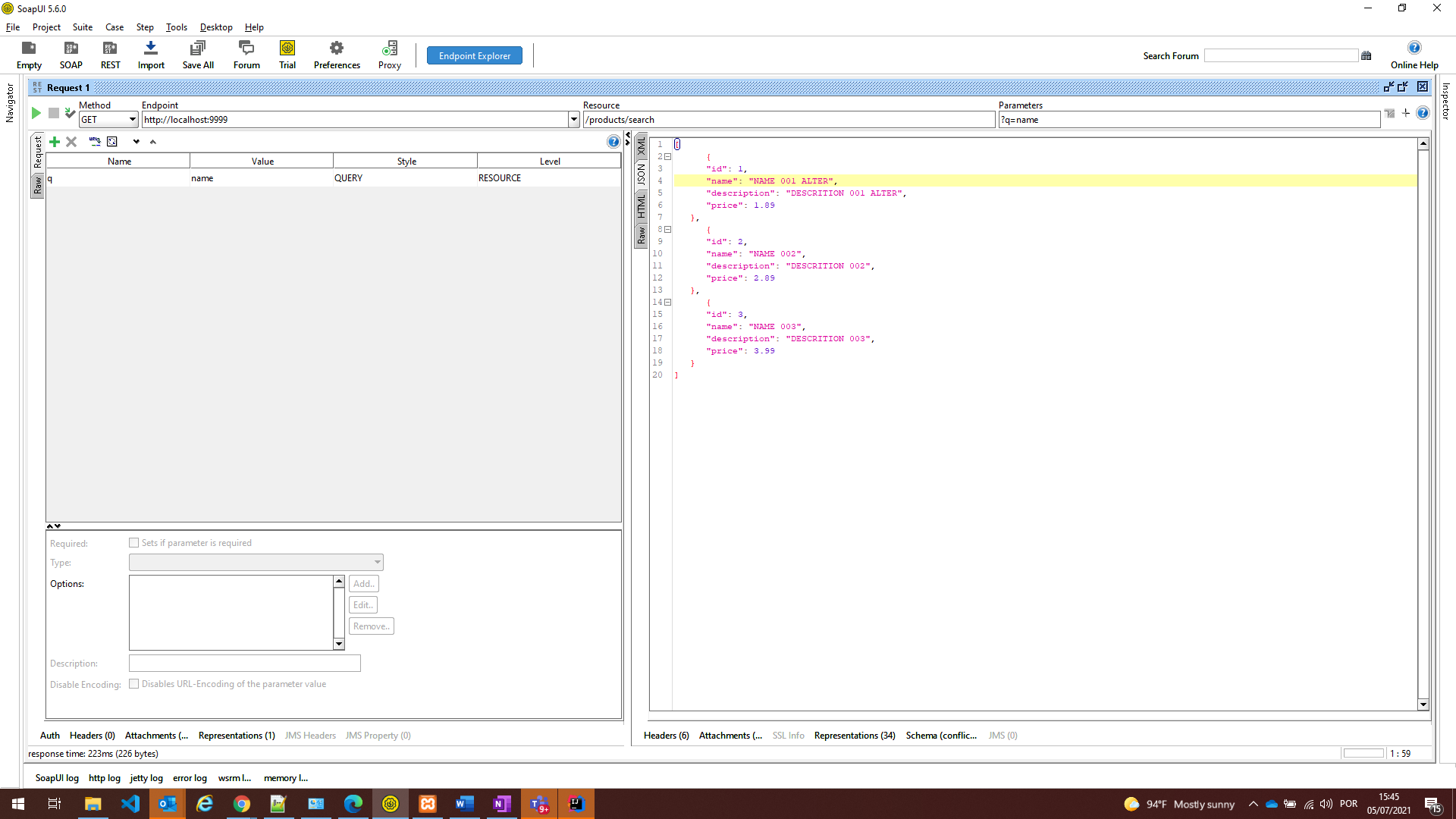


#### Execução do teste manual

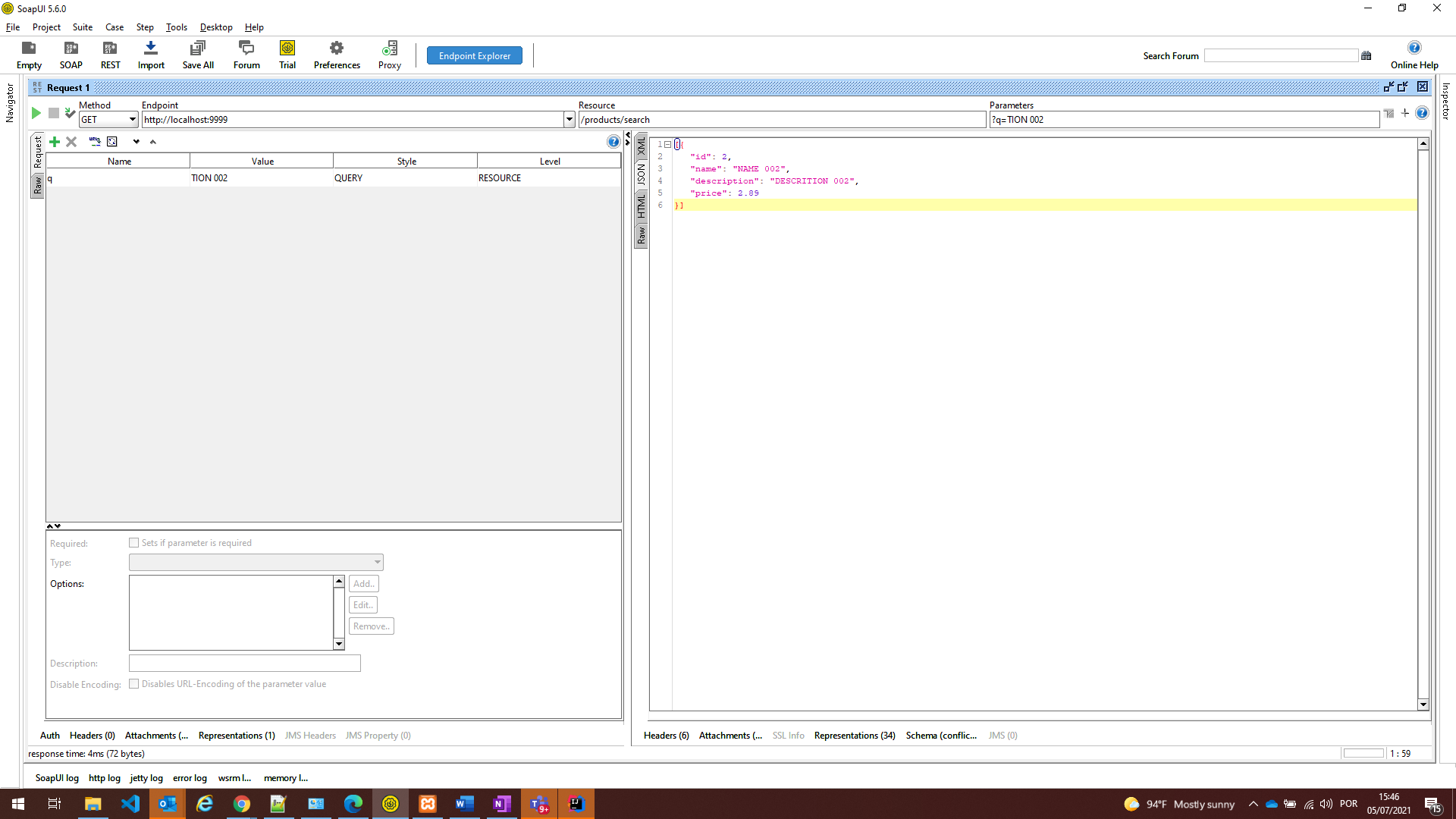
* Print do banco de dados no momento do teste.



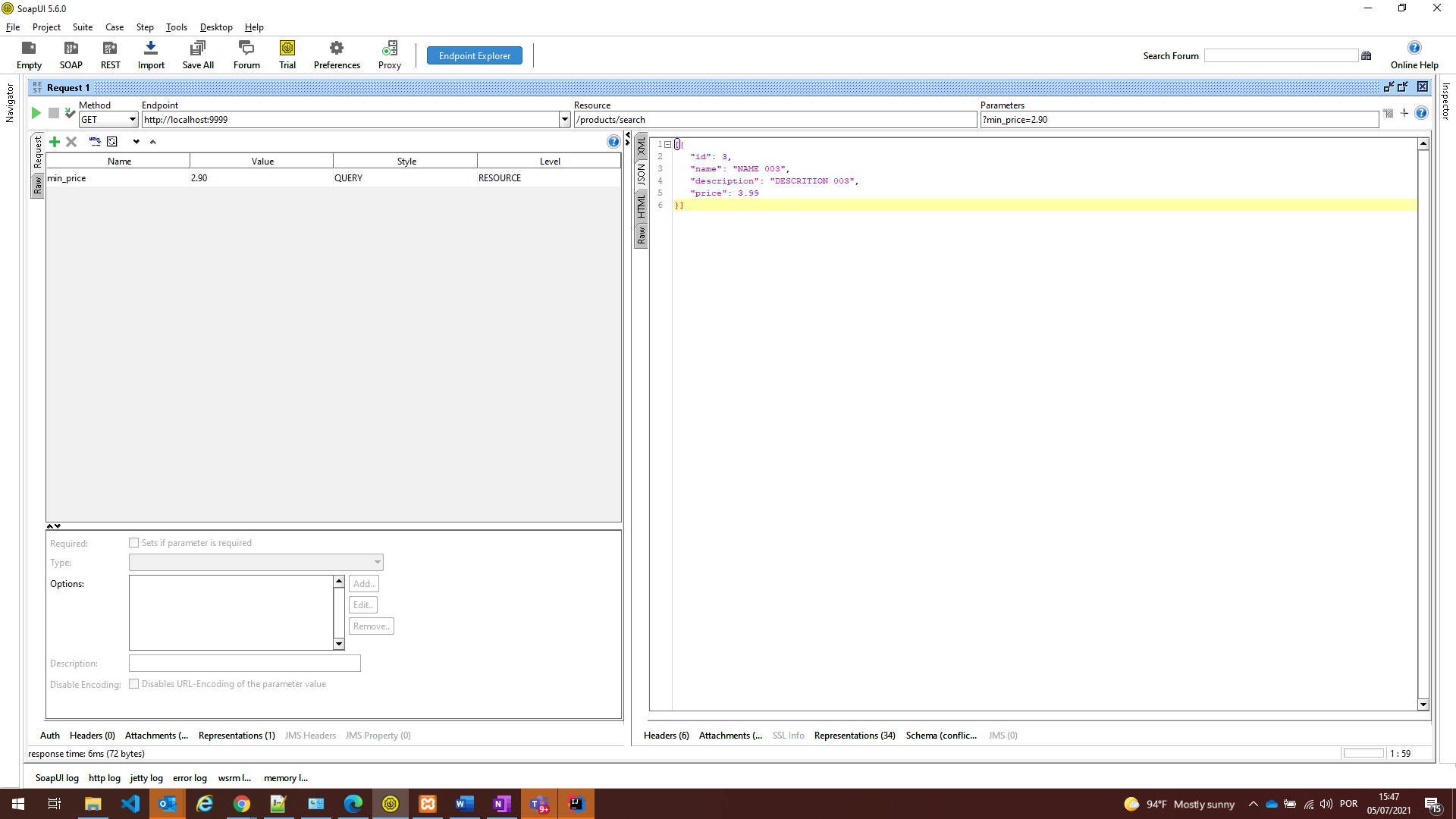
* Consulta pelo name.



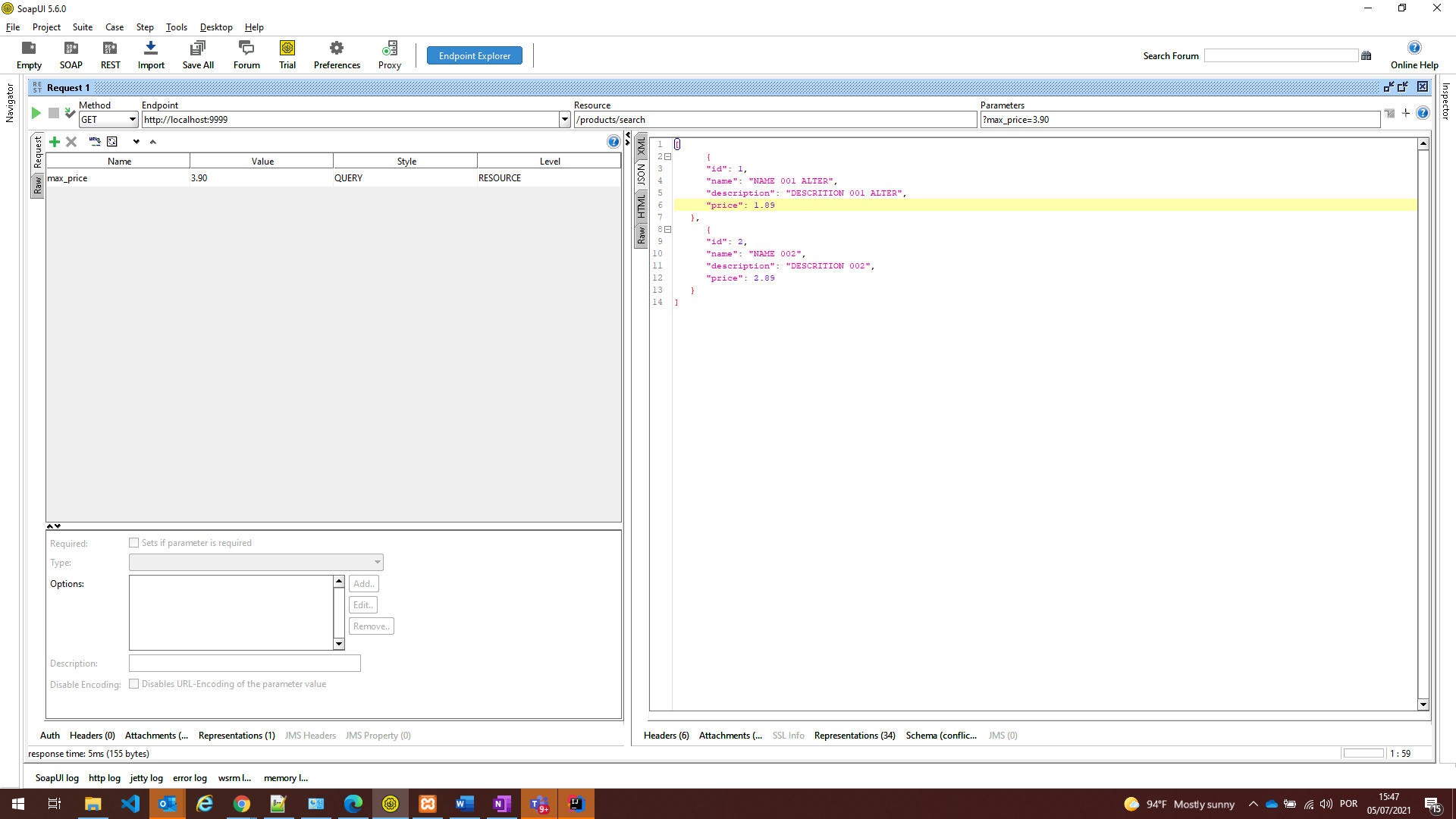
* Consulta pela description.



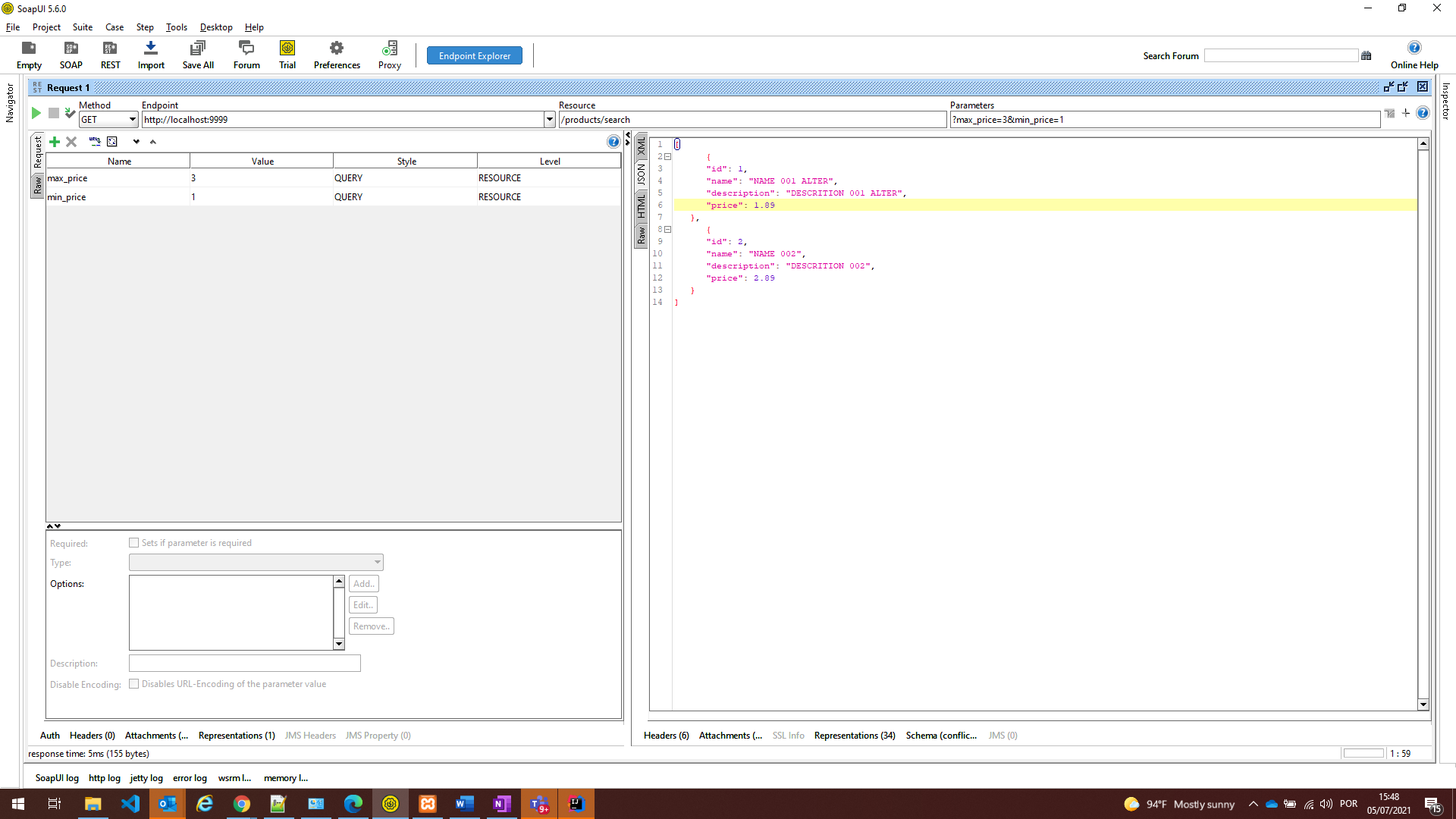
* Consulta pelo price (min\_price).



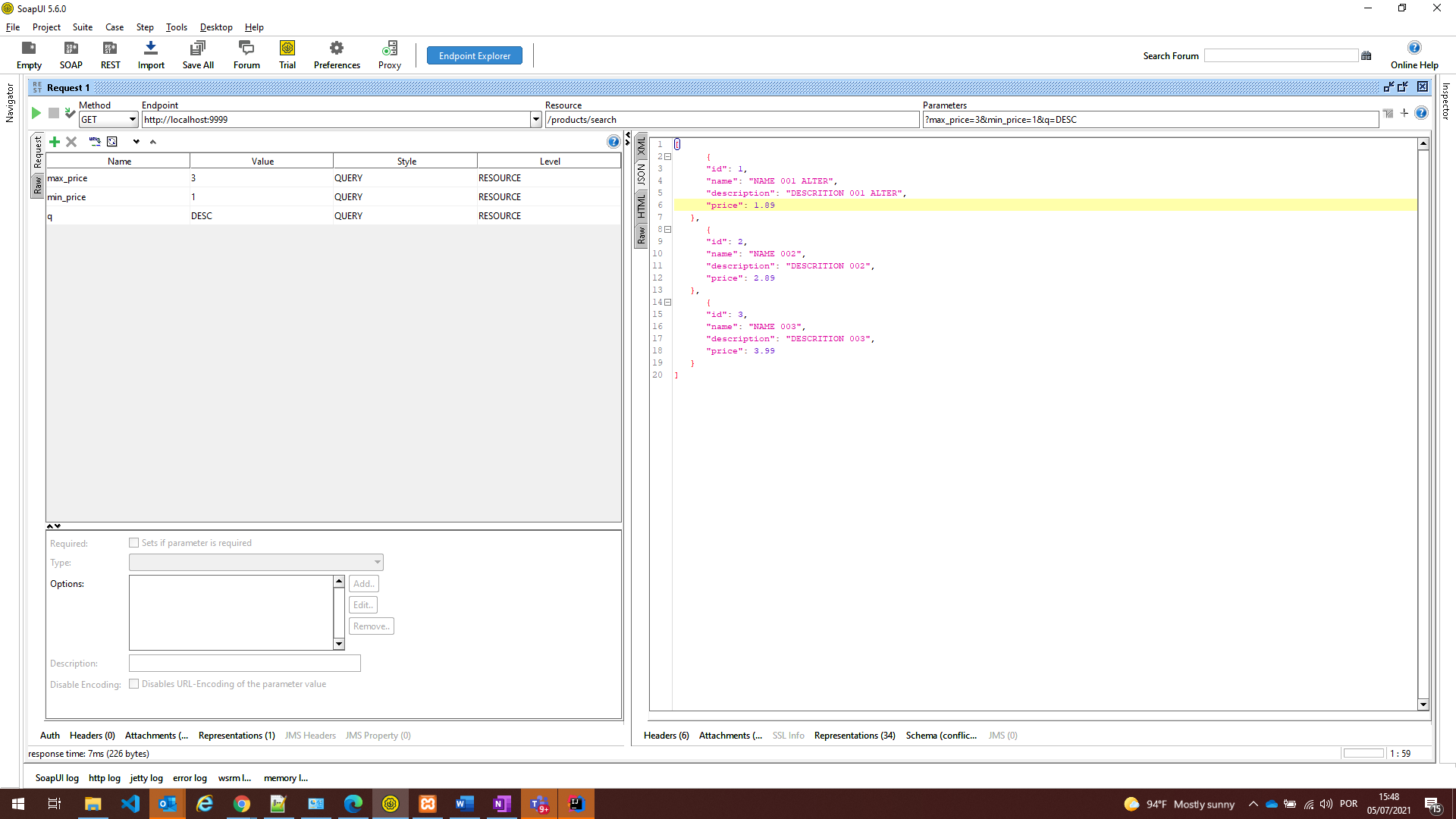
* Consutla pelo price (max\_price).



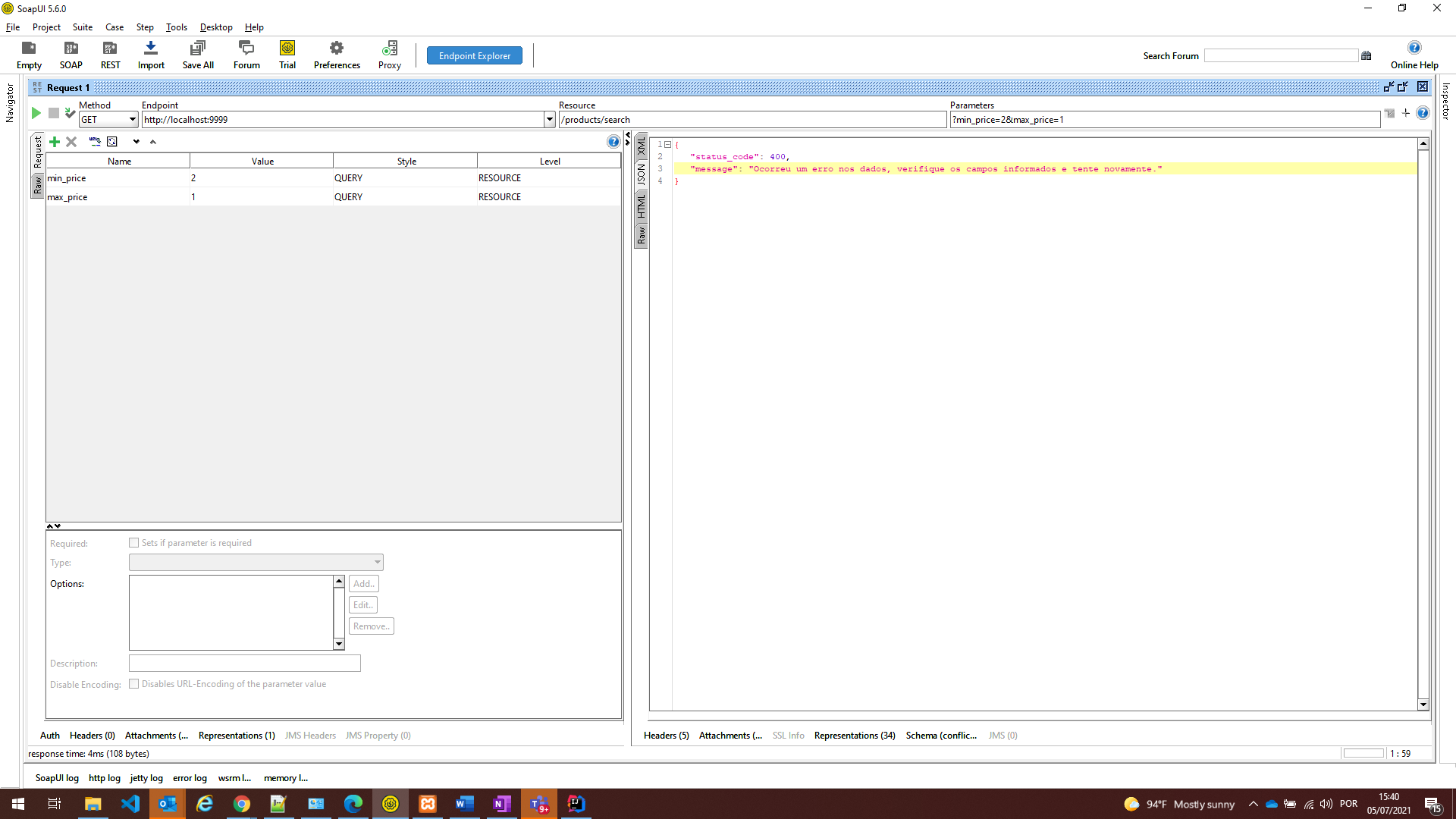
* Consulta passando apenas o price (min\_price e max\_price).



* Consulta utilizando todos os parâmetros.



* Validação de price (min\_price > max\_price).



### deleteProduto – [DELETE]

#### Descrição

Esse endpoint deleta um registro de produto na base de dados de acordo com o id do produto informado.

Caso o microserviço encontre o produto filtrando pelo id então ele realiza o delete e retornar um HTTP 200.

Se o id informado não for localizado, o microserviço retornar um HTTP 404.

* **Link localhost:** 
  + [**http://localhost:9999/products/{id}**](http://localhost:9999/products/%7bid%7d)

#### Assinatura do Método

public void deleteProduto(@PathVariable Long id)

#### Parâmetros de Entrada

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |

#### Retorno do Método

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| void | Não possui retorno. |

**Objeto - DesafioCompasso (Produto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Ordem** | **Nome** | **Tipo** | **Descrição** |
| 1 | id | Long | Id do produto. |
| 2 | name | String | Nome do produto. |
| 3 | description | String | Descrição do produto. |
| 4 | price | Double | Preço do produto. |

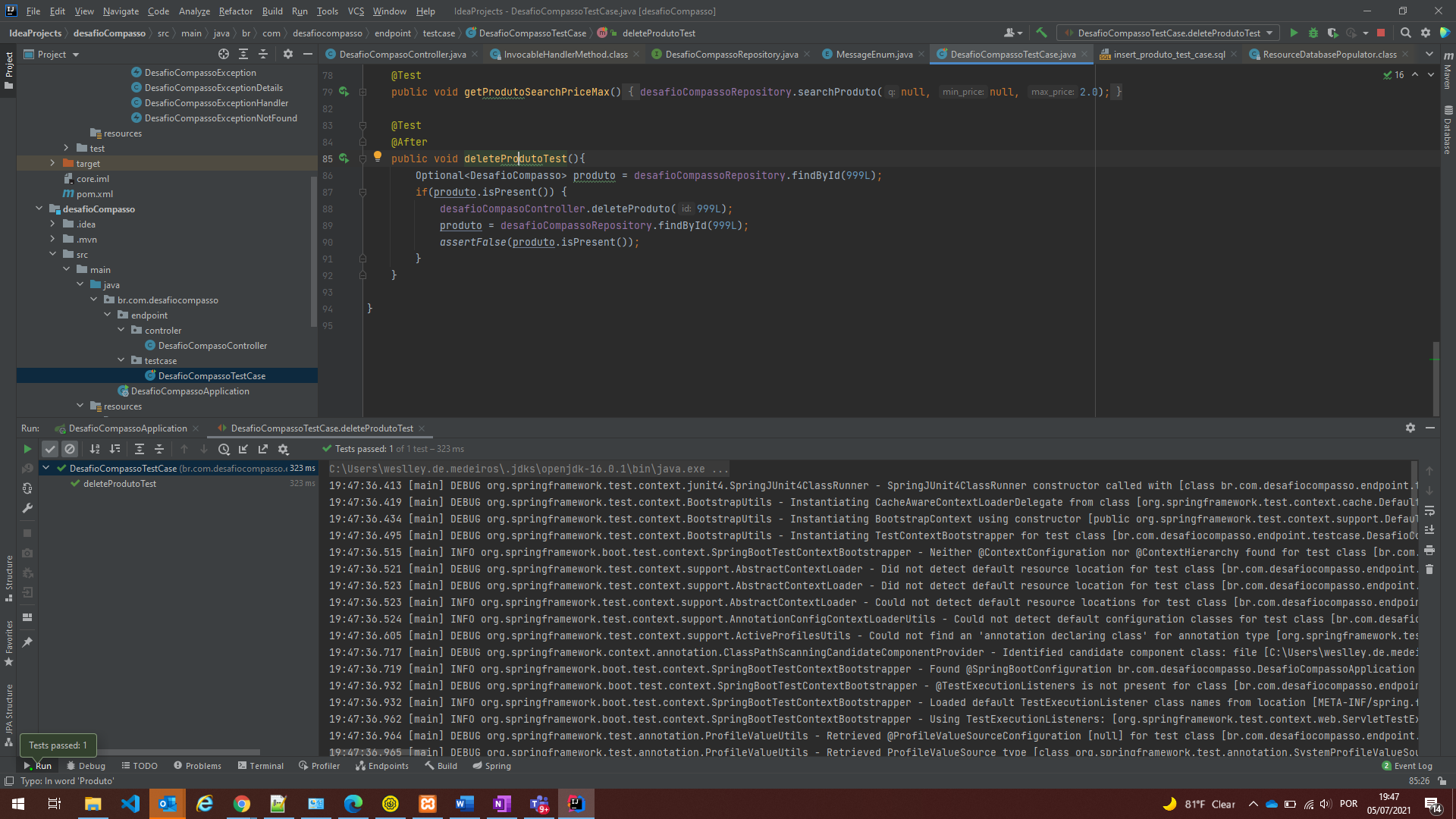
**JSON – Status 404:**

|  |
| --- |
| {  "status\_code": 404,  "message": "Produto com ID: 99 inexistente."  } |

#### Regras de Negócios

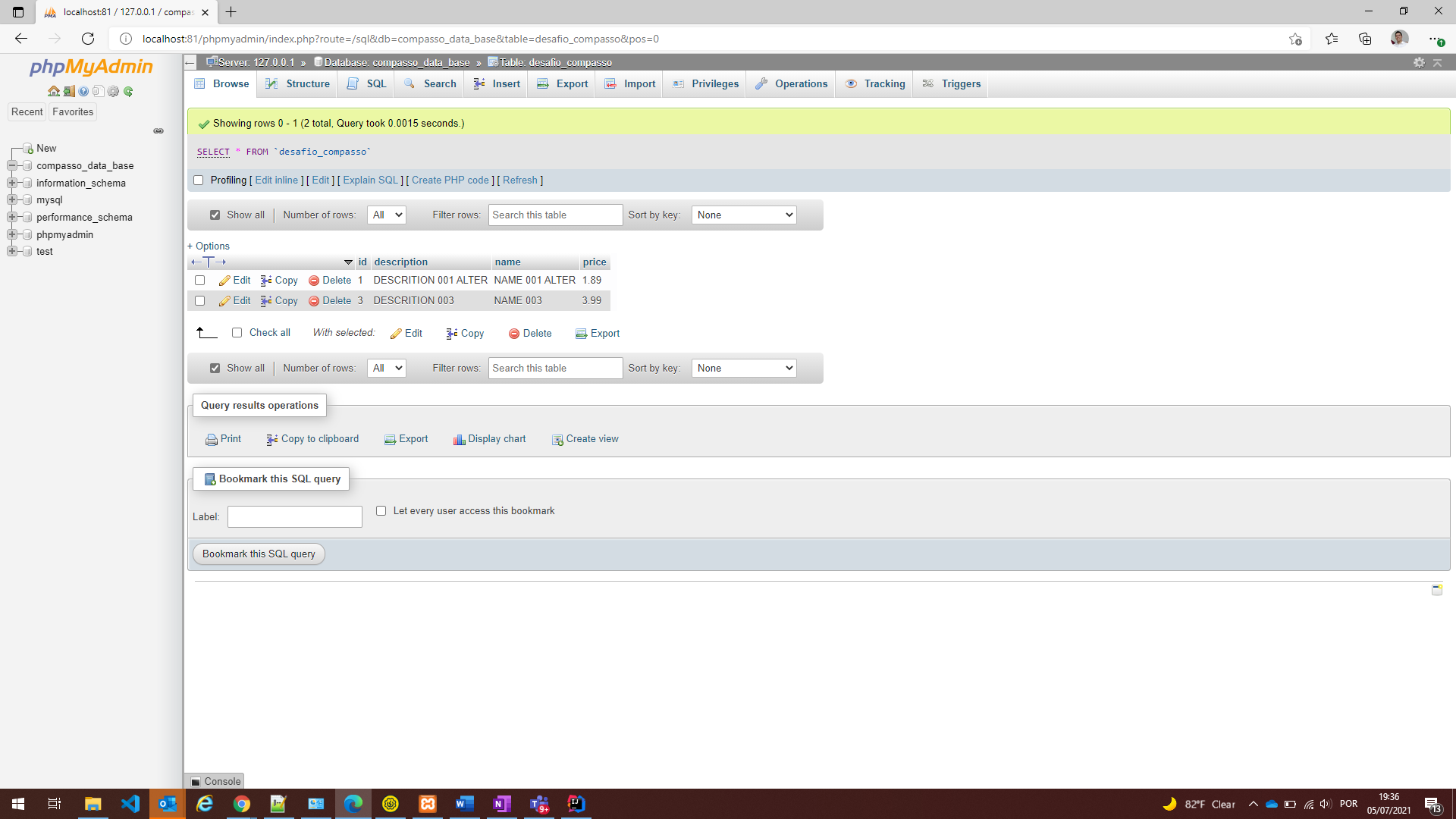
* Caso o microserviço identifique o id do produto informado, o mesmo será deletado da base de dados e retornará o HTTP 200.
* Caso o id informado não exista na base de dados é retornado o JSON conforme informado acima com o HTTP404.

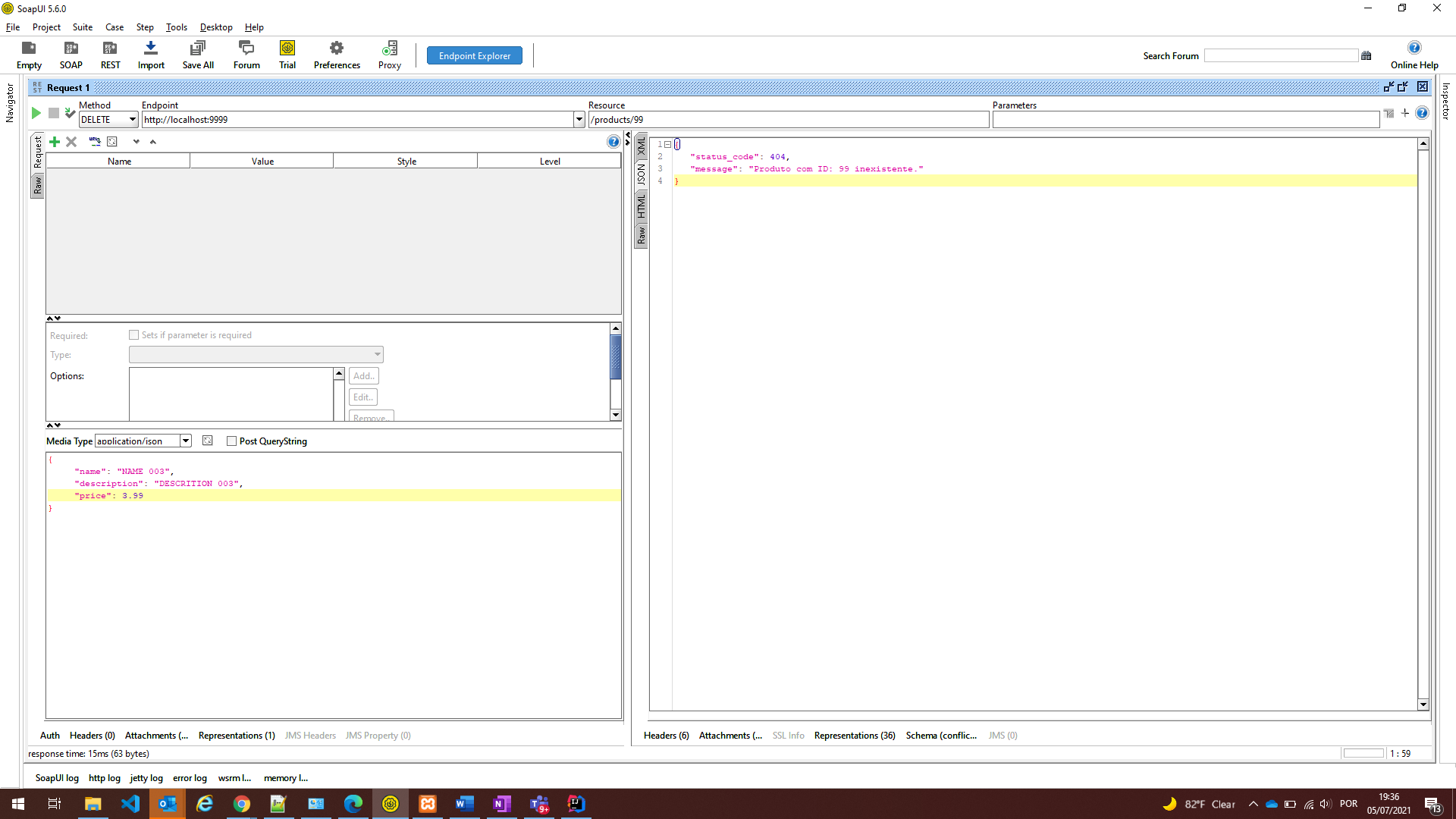
#### Execução do teste JUnit



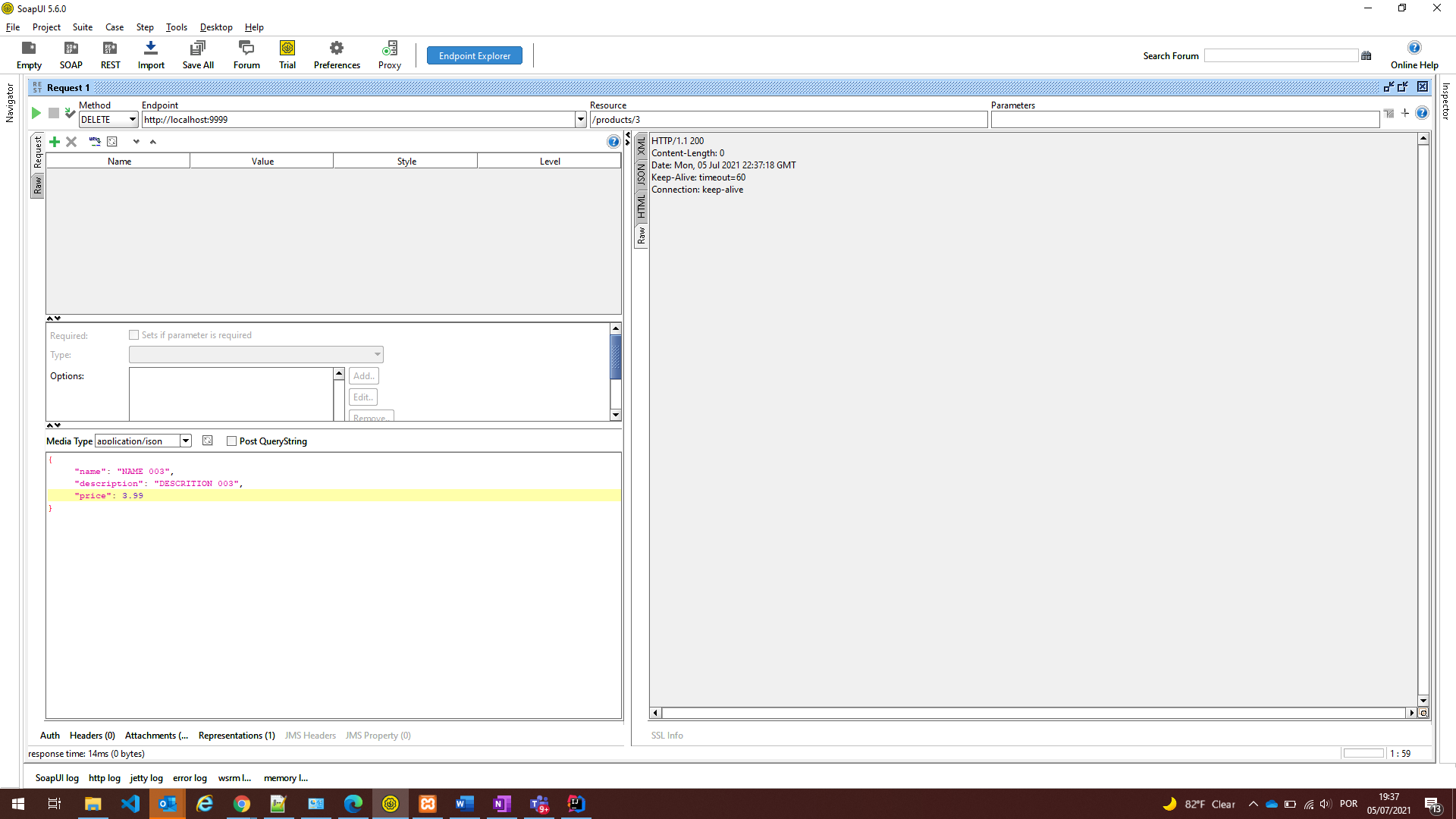
#### Execução do teste manual

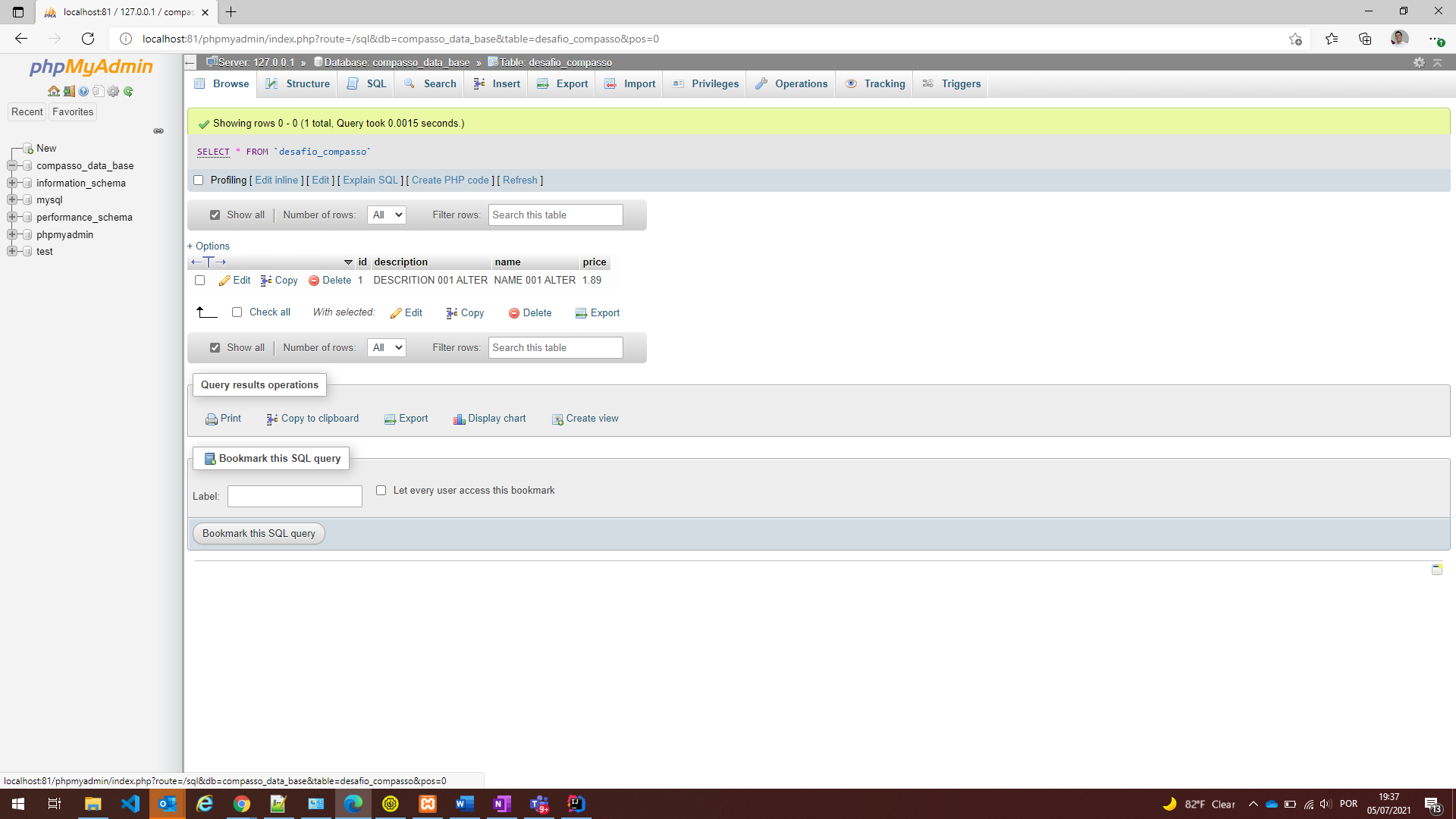
* Passando Id inexistente na base de dados:





* Passando Id válido.





## Erros e Mensagens

|  |  |  |
| --- | --- | --- |
| **Código** | **Descrição** | **Motivo** |
| **201** | O status HTTP "201 Created" é utilizado como resposta de sucesso, indica que a requisição foi bem sucedida e que um novo recurso foi criado | O cadastro do produto foi realizado com sucesso. |
| **200** | O código HTTP 200 OK é a resposta de status de sucesso que indica que a requisição foi bem sucedida. | A requisição foi realizada com sucesso. |
| **400** | O código de status de resposta HTTP 400 Bad Request indica que o servidor não pode ou não irá processar a requisição devido a alguma coisa que foi entendida como um erro do cliente. | Sintaxe de requisição mal formada, enquadramento de mensagem de requisição inválida ou requisição de roteamento enganosa |
| **404** | O código 404 ou 4:04 é um código de resposta HTTP que indica que o cliente pôde comunicar com o servidor, mas o servidor não pôde encontrar o que foi pedido, ou foi configurado para não cumprir o pedido e não revelar a razão, a página não existe mais ou a URL foi inserida correctamente. | Id do produto informado é inexistente. |

## Execução NUnit completa:

