

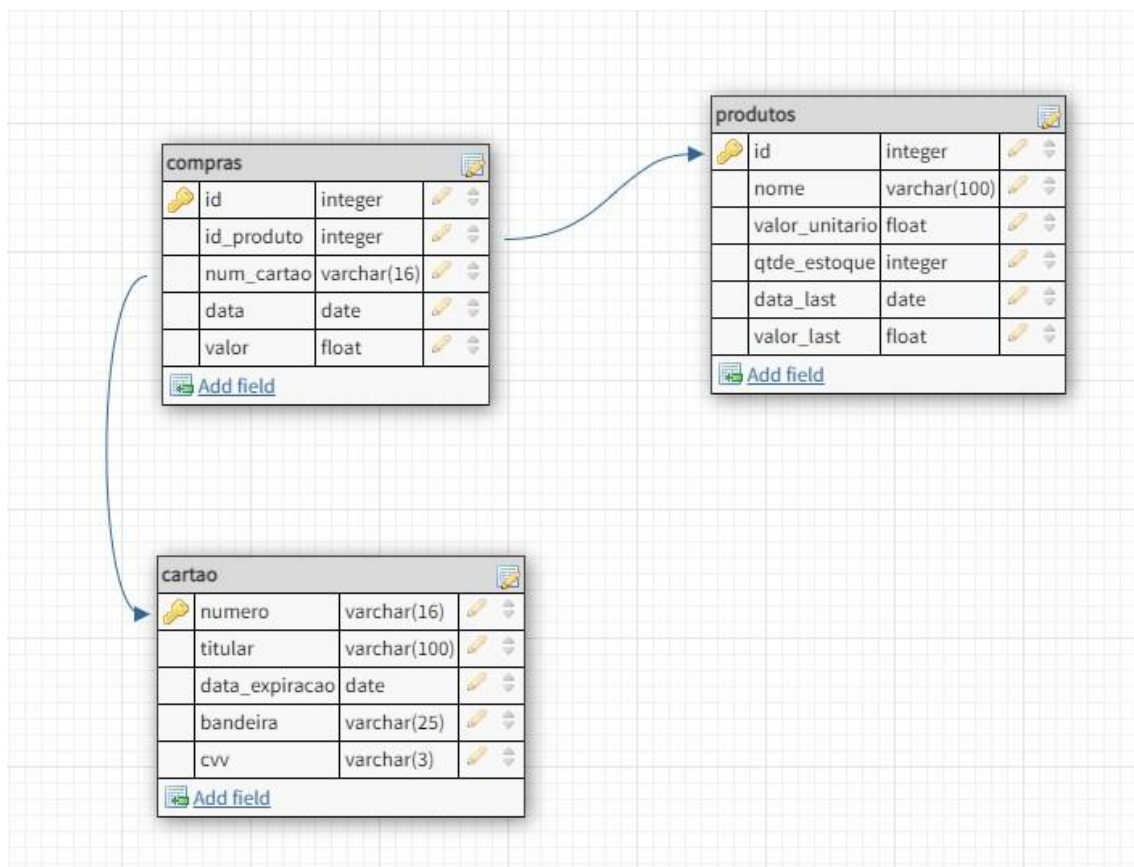
Raphael API – Estudo de Caso e Documentação

1. Introdução e Breve Estudo de Caso

Raphael API é uma API REST desenvolvida com a plataforma .NetCore e Entity Framework, serve de backend de um sistema de registro de produtos e compras e controle de estoque.

Possui dois endpoints: “/api/produtos” e “/api/compras”, a API de pagamentos possui um, “/api/pagamento/compras”, mas esta será discutida mais adiante, à parte.

As principais entidades são: Produto, Cartao e Compra, relacionando-se conforme o modelo:



Este modelo representa o Banco de Dados gerenciado pela API, as seguintes consultas ao Banco de Testes demonstram:

```
1 select * from produto order by id;
```

Id	nome	valor_unitario	qtde_estoque	data_last	valor_last
1	Teclado USB Multilaser	34,58	3	2021-06-02 00:06:20.6759017	103,74
2	Notebook DELL A432	1542,9	4	2021-06-02 01:25:41.3941992	12343,2
3	Pendrivel Kingston 32GB	52,65	15	NULL	0
4	Processador Intel Core I7 9900K	3546,65	23	NULL	0

```
1 select * from compra order by id;
```

Id	produto_Id	cartaonumero	valor	qtde_comprada
1	1	3322556678945612	69,16	2
2	2	9988445577661122	103,74	3
3	3	5544221265985247	12343,2	8

```
1 select * from cartao order by numero;
```

numero	data_expiracao	bandeira	cvv	titular
3322556678945612	12/2024	MASTERCARD	554	Castro Diaz
5544221265985247	06/2027	VISA	852	Maria Cajé
9988445577661122	04/2024	MASTERCARD	774	Glória Tereza

Essas tabelas são baseadas nas Classes Modelos do Projeto, encontradas na pasta “/Models”, definidas em “Produto.cs”, “Cartao.cs”, “Compra.cs”.

Além disso, na pasta “/DTO” são definidas as classes Data Transfer Object, que possuem dados mais enxutos que evitam o excesso de tráfego de informações

desnecessárias, como numero do cartão de crédito, etc. que ficam acessíveis apenas pelo próprio Banco de Dados. São duas classes DTO’s: “ProdutoDTO.cs” e “CompraDTO.cs”.

Requisições:

O endpoint “/api/produtos” é controlado por ProdutosController, onde são atendidas as seguintes requisições por funções:

POST	AdicionarProduto()
GET	ListarProdutos()
GET/id	DetalharProduto()
DELETE	DeletarProduto()

Além do método ModificarProduto() que será tratado mais adiante.

Ao fazer GET em “/api/produtos”, se obtém uma lista de ProdutoDTO com os produtos registrados no Banco de Dados.

The screenshot shows a REST client interface with a GET request to `https://localhost:8080/api/produtos/`. The response is a JSON array of four product objects. The status is 200 OK, time is 16 ms, and size is 450 B.

```
1 {
2   {
3     "nome": "Teclado USB Multilaser",
4     "valor_unitario": 34.58,
5     "qtde_estoque": 3
6   },
7   {
8     "nome": "Mouse Óptico USB Multilaser",
9     "valor_unitario": 44.9,
10    "qtde_estoque": 41
11  },
12  [
13    "nome": "Notebook DELL A432",
14    "valor_unitario": 1542.9,
15    "qtde_estoque": 12
16  ],
17  {
18    "nome": "Pendrive Kingston 32GB",
19    "valor_unitario": 52.65,
20    "qtde_estoque": 15
21  }
22 }
```

Ao fazer POST do JSON compatível com os atributos da classe ProdutoDTO o usuário aguarda que se registre o novo produto no banco de dados, obtendo o código de status 201 e o JSON do objeto Produto registrado.

POST ▼ https://localhost:8080/api/produtos/ Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {
2   "nome": "Processador Intel Core I7 9900K",
3   "valor_unitario": 3546.65,
4   "qtde_estoque": 23
5 }
```

Body Cookies Headers (5) Test Results Status: 201 Created Time: 172 ms Size: 326 B

Pretty Raw Preview Visualize **JSON** ⌵

```
1 {
2   "id": 5,
3   "nome": "Processador Intel Core I7 9900K",
4   "valor_unitario": 3546.65,
5   "qtde_estoque": 23,
6   "data_last": null,
7   "valor_last": 0
8 }
```

Ao verificar a listagem novamente:

GET ▼ https://localhost:8080/api/produtos/

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK Time: 13 ms

Pretty Raw Preview Visualize **JSON** ⌵

```
1 [
2   {
3     "nome": "Teclado USB Multilaser",
4     "valor_unitario": 34.58,
5     "qtde_estoque": 3
6   },
7   {
8     "nome": "Mouse Óptico USB Multilaser",
9     "valor_unitario": 44.9,
10    "qtde_estoque": 41
11  },
12  {
13    "nome": "Notebook DELL A432",
14    "valor_unitario": 1542.9,
15    "qtde_estoque": 12
16  },
17  {
18    "nome": "Pendrive Kingston 32GB",
19    "valor_unitario": 52.65,
20    "qtde_estoque": 15
21  },
22  {
23    "nome": "Processador Intel Core I7 9900K",
24    "valor_unitario": 3546.65,
25    "qtde_estoque": 23
26  }
27 ]
```

Ao passar o {id} à requisição GET, retorna o objeto Produto que possui o mesmo id:

GET

https://localhost:8080/api/produtos/1

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body	Cookies	Headers (4)	Test Results	Status: 200 OK
Pretty	Raw	Preview	Visualize	JSON
<pre>1 { 2 "id": 1, 3 "nome": "Teclado USB Multilaser", 4 "valor_unitario": 34.58, 5 "qtde_estoque": 3, 6 "data_last": "2021-06-02T00:06:20.6759017", 7 "valor_last": 103.740005 8 }</pre>				

No endpoint “/api/compras”, O usuário requisita o registro de uma compra por meio de um POST do JSON compatível com objeto de classe CompradDTO, no caso da API Pagamentos recusar, ou a quantidade em estoque ser insuficiente a compra não se sucede.

POST

https://localhost:8080/api/compras/

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

7

8

9

10

11

```
"produto_Id":1,
"cartao":{
  "numero":"9988445577661122",
  "titular":"Glória Tereza",
  "cvv":"774",
  "data_expiracao":"04/2024",
  "bandeira":"MASTERCARD"
},
"qtde_comprada": 1
```

Body

Cookies

Headers (4)

Test Results

Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

```
"type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
"title": "Bad Request",
"status": 400,
"traceId": "00-f83129d5d0a07d4face29753d9d2fe0b-7e11df6baebec046-00"
```

Neste caso acima, a API Pagamentos rejeitou a compra (valor da compra ≤ 100).

POST <https://localhost:8080/api/compras/>

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼

```
1 {
2   "produto_Id": 3,
3   "cartao": {
4     "numero": "5544221265985247",
5     "titular": "Maria Cajé",
6     "cvv": "852",
7     "data_expiracao": "06/2027",
8     "bandeira": "VISA"
9   },
10  "qtde_comprada": 8
11 }
```

Body Cookies Headers (4) Test Results Status: 200 OK T

Pretty Raw Preview Visualize JSON ▼

```
1 {
2   "id": 3,
3   "produto_Id": 3,
4   "cartao": {
5     "numero": "5544221265985247",
6     "data_expiracao": "06/2027",
7     "bandeira": "VISA",
8     "cvv": "852",
9     "titular": "Maria Cajé"
10  },
11  "valor": 12343.2,
12  "qtde_comprada": 8
13 }
```

Neste acima, a compra obtém sucesso, e vê-se a diminuição do estoque do produto comprado abaixo:

GET <https://localhost:8080/api/produtos/>

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON ▼

```
1 [
2   {
3     "nome": "Teclado USB Multilaser",
4     "valor_unitario": 34.58,
5     "qtde_estoque": 3
6   },
7   {
8     "nome": "Notebook DELL A432",
9     "valor_unitario": 1542.9,
10    "qtde_estoque": 4
11  },
12  {
13    "nome": "Pendrive Kingston 32GB",
14    "valor_unitario": 52.65,
15    "qtde_estoque": 15
16  },
17  {
18    "nome": "Processador Intel Core I7 9900K",
19    "valor_unitario": 3546.65,
20    "qtde_estoque": 23
21  }
22 ]
```

Há uma verificação quanto ao número do cartão inválido, por tamanho ou caractere indevido:

```
1 {
2   "produto_Id":3,
3   "cartao":{
4     "numero":"4511454545454c",
5     "titular":"Maria Cajé",
6     "cvv":"852",
7     "data_expiracao":"06/2027",
8     "bandeira":"VISA"
9   },
10  "qtde_comprada": 1
11 }
```


Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON 

```
1 {
2   "status": 412,
3   "traceId": "00-765bb24a864df94a98459ab966999c6f-e0fe1b5432dbae47-00"
4 }
```

```
1 {
2   "produto_Id":3,
3   "cartao":{
4     "numero":"451145454545454",
5     "titular":"Maria Cajé",
6     "cvv":"852",
7     "data_expiracao":"06/2027",
8     "bandeira":"VISA"
9   },
10  "qtde_comprada": 1
11 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON 

```
1 {
2   "status": 412,
3   "traceId": "00-ff2fa72908225c4a9ab3896a272794f9-04ebfb7253782547-00"
4 }
```

2. Documentação - Funções e Métodos Internos:

2.1 RaphaelAPI.Models

2.1.1 Produto

Possui os seguintes atributos:

- `public int` Id, que é a chave primária do produto;
- `public string` nome, que é o nome do produto;
- `public float` valor_unitario, que é o valor de cada produto;
- `public int` qtde_estoque, que é a quantidade em estoque;
- `public Nullable<DateTime>` data_last, que é a data da ultima compra feita deste produto, podendo ser null;
- `public float` valor_last, que é o valor desta ultima compra.

Possui um construtor padrão vazio e um construtor que recebe um ProdutoDTO como argumento.

Métodos:

- `public ProdutoDTO ProdutoToDTO ()`, retorna um ProdutoDTO baseado na instancia de Produto;

2.1.2 Compra

Possui os seguintes atributos:

- `public int` Id, que é a chave primária da compra;
- `public int` produto_Id, que é a chave estrangeira do produto comprado;
- `public` Cartao cartão, objeto Cartao para pagamento;
- `public float` valor, valor da compra, é calculado pelo método RealizarCompra;
- `public int` qtde_comprada, quantidade a ser comprada;

Possui os métodos:

- `public int` RealizarCompra(`ref` Produto produto), recebe um Produto por referência, verifica se possui estoque disponível para compra, modifica o objeto produto com a nova qtde_estoque subtraída pela qtde_comprada, atualiza data_last e valor_last do produto, dá baixa no estoque, retorna 0 se bem sucedida, ou -1 se o estoque for insuficiente.
- `public bool` IsValid(), retorna true se o objeto Compra é inválido, verificando, inclusive, a validade do cartão;

2.1.3 Cartao

Possui os seguintes atributos:

- `public string` numero, que é o número e chave primária do cartão;
- `public string` data_expiracao, autoexplicativo;
- `public string` bandeira, autoexplicativo;
- `public string` cvv, autoexplicativo;
- `public string` titular, autoexplicativo;

Metodos:

- `public bool` IsInvalid(), retorna true se o objeto Cartao é inválido;

2.2 RaphaelAPI.DTO

2.2.1 ProdutoDTO

Possui os seguintes atributos:

- `public string` nome, mesmo que seu homônimo em Produto(2.1.1);
- `public float` valor_unitario, mesmo que seu homônimo em Produto(2.1.1);
- `public int` qtde_estoque, mesmo que seu homônimo em Produto(2.1.1);

Possui um construtor padrão vazio e outro que recebe por argumento um objeto Produto.

Métodos:

- `public` Produto DT0toProduto (), retorna um objeto Produto baseado no ProdutoDTO;
- `public bool` IsInvalid(), retorna true se o ProdutoDTO não for válido;

2.2.2 CompraDTO

Possui os seguintes atributos:

- `public int` Id, mesmo que seu homônimo em Compra(2.1.2);
- `public int` produto_Id, mesmo que seu homônimo em Compra(2.1.2);
- `public float` valor, mesmo que seu homônimo em Compra(2.1.2);

Possui um construtor padrão vazio e outro que recebe um objeto Compra por argumento;

2.3 RaphaelAPI.Controllers

2.3.1 ProdutosController

Controla as requisições em “api/produtos”.

Atributos:

- `private readonly ApiContext _context`, contexto para o Banco de Dados;

Possui um único construtor, o padrão, que inicializa `_context`.

Métodos(Tratamento de Requisições):

- `public async Task<ActionResult<List<ProdutoDTO>>> ListarProdutos()`, responde a solicitações HTTP GET, retorna uma lista com todos produtos no BD como objetos ProdutoDTO, no caso de sucesso retorna status 200, em caso de falha retorna 400;
- `public async Task<ActionResult<Produto>> DetalharProduto(int id)`, responde ao GET com {id}, (“/api/produtos/{id}”), se sucesso, retorna objeto Produto com status 200, se falhar por não encontrar produto com id no BD, retorna 404, outras falhas, 400;
- `public async Task<ActionResult<Produto>> AdicionarProduto(ProdutoDTO produto)`, responde ao POST de objeto ProdutoDTO, se o produto não tiver recebido argumentos válidos, retorna status 412, caso obtenha sucesso, retorna um Produto com status 200;
- `protected async Task<IActionResult> ModificarProduto(int id, Produto produto)`, função interna a API, modifica dados de produto baseado no id, salvando modificações no BD, no caso de sucesso, nada retorna, no caso de não encontrar o produto, retorna status 404, e no caso de falha, retorna status 400;
- `public async Task<IActionResult> DeletarProduto(int id)`, responde à requisição HTTP DELETE, deleta o produto com o mesmo id do banco de dados, No caso de não encontrar, retorna status 412, no caso de falhas, 400, no caso de sucesso, nada retorna;
- `private bool ProdutoExists(int id)`, retorna verdadeiro se já houver algum produto no BD com o id;

2.3.2 ComprasController

Controla as requisições em “api/produtos”.

Atributos:

- `private readonly` `ApiContext _context`, contexto para o Banco de Dados;

Possui um único construtor, o padrão, que inicializa `_context`.

Métodos(Tratamento de Requisições):

- `public async Task<ActionResult<Compra>> FazerCompra(Compra compra)`, realiza a compra a uma requisição POST, passando id do produto, a quantidade e o objeto `Cartao` necessariamente, a função então, verifica se o cartão é válido, verifica se o produto existe, chama o método `RealizarCompra` (2.1.2), inicia uma requisição POST na API Pagamentos que responde um objeto `Pagamento` com campo estado “APROVADO” ou “REJEITADO”, no primeiro caso, atualiza o Banco de Dados, deletando linhas caso não haja mais produtos em estoque, atualizando e retornando o objeto compra detalhado e status 200; no caso “REJEITADO” ou de falha, retorna status 400;

2.4 RaphaelAPI.Data

2.4.1 ApiContext

É a classe de contexto para o Entity Framework, define as tabelas do Banco de Dados em relação as Classes do projeto:

- `public` `DbSet<Produto>` `produto`;
- `public` `DbSet<Compra>` `compra`;
- `public` `DbSet<Cartao>` `cartão`;

3. API Pagamentos

3.1 RaphaelAPI.Pagamentos

Esta API é criada com a intenção de simular uma interface que aprova ou rejeita solicitações de pagamentos por cartão de crédito.

No estado atual, ela apenas retorna estado “APROVADO”, se o valor da compra for maior que 100, e “REJEITADO”, se o valor for menor ou igual a 100.

3.2 RaphaelAPI.Pagamentos.Models

3.2.1 Pagamento

Classe modelo para resposta às requisições de pagamentos.

Atributos:

- `public float` valor, valor requisitado;
- `public string` estado, "APROVADO" ou "REJEITADO";

Possui um construtor padrão vazio e um construtor com dois argumentos `public Pagamento(float valor, string estado)`.

3.3 RaphaelAPI.Pagamentos.Controllers

3.3.1 PagamentosController

Responde às requisições POST em `"/api/pagamento/compras/"`.

Métodos:

- `public Pagamento FazerPagamento(Compra compra)`, trata POST de compra, verifica o valor, se maior que 100, retorna Pagamento com estado "APROVADO", caso contrário, "REJEITADO";