Machine Learning Engineer

Python para ML e IA

Desenvolvimento de API com FastAPI

Leonardo Pena

/ Seja muito bem vindo

FOCO

Criar CRUD de itens usando BaseModel e rotas GET/POST/PUT/DELETE

REVISÃO

Revisaremos como o FastAPI retorna automaticamente JSON de um objeto Python

EXEMPLO

Exemplos de tratamento de exceções com HTTPException

Objetivo dessa primeira parte

Passo 1

Apresentar a classe Item(BaseModel) e mostrar como funciona Pydantic no FastAPI

Passo 2

Implementar rotas CRUD para gerenciar uma lista de itens em memória

Passo 3

Explicar uso de status codes como 201, 404, etc.

Passo 4

Mostrar o manuseio de erros via HTTPException

Passo 5

Preparar terreno para o scraping

Criação de API FastAPI

/ Definindo a Classe Item

- BaseModel do Pydantic permite criar modelos de dados tipados
- Campos name, description, price, quantity
- Se enviarmos dados inválidos, FastAPI gera resposta 422 (Unprocessable Entity)
- Diminui boilerplate e checagens manuais
- Ajuda no autocompletar e doc das rotas

/ Mas antes

- Instalamos *pydantic*
- importamos
  - from pydantic import BaseModel

/

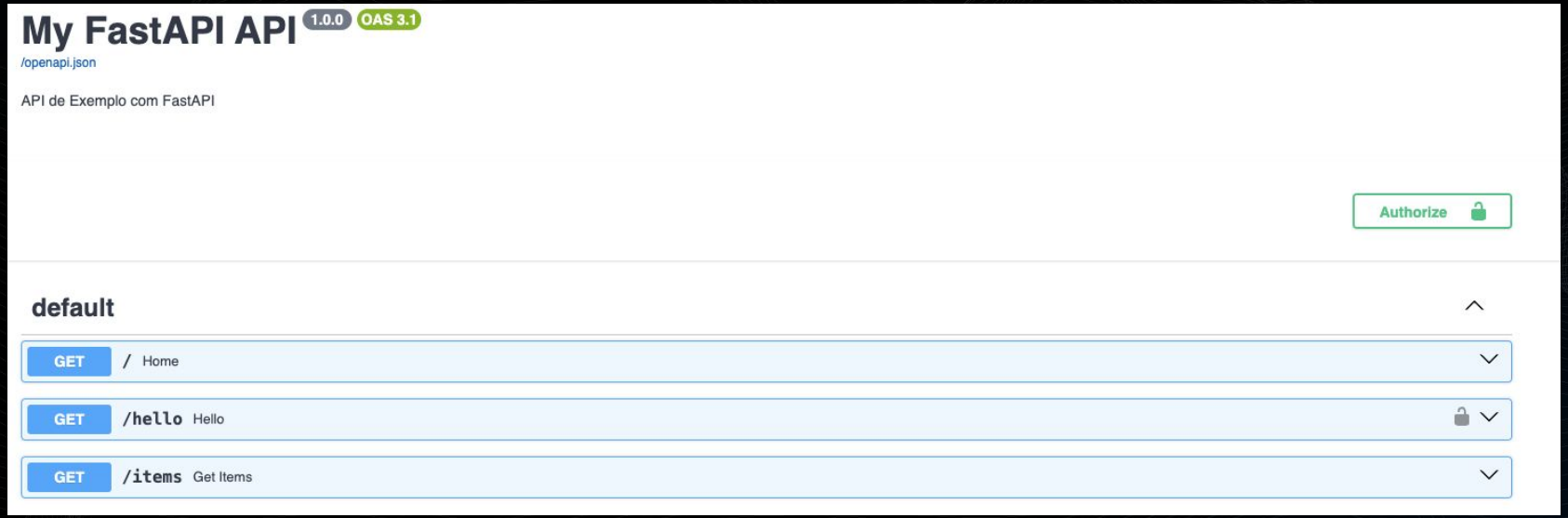
Classe Item

```python
from pydantic import BaseModel

# Lista de itens em memória para operações CRUD
items = []  # armazena itens como dicionários

class Item(BaseModel):
    name: str             # nome do item
    description: str = None  # descrição opcional
    price: float = None   # preço opcional
    quantity: int = None  # quantidade opcional
```

/

Rota GET /items (Leitura de Todos)

```python
@app.get("/items")
async def get_items():
    return items
```

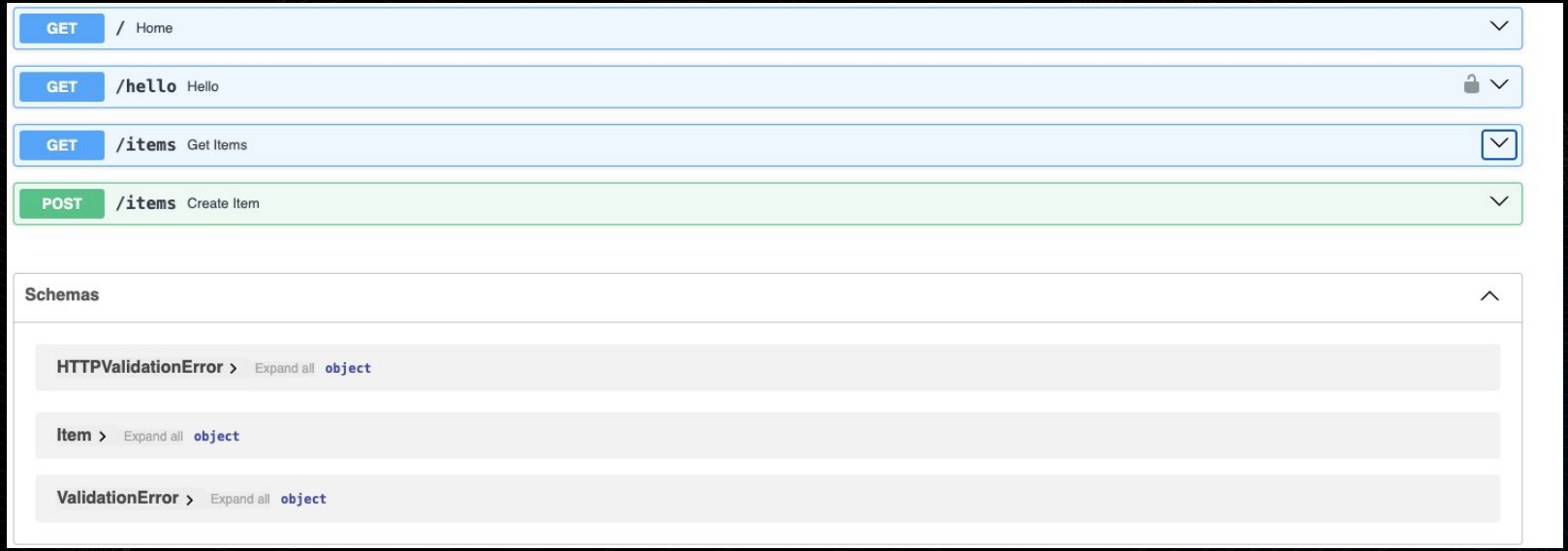


/

Rota POST /items (Criação)

```python
@app.post("/items", status_code=201)
async def create_item(item: Item):
    items.append(item.dict())
    return item
```

Vamos testar mais abaixo



/

Rota PUT /items/{item_id} (Atualização)

```python
@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item):
    if 0 <= item_id < len(items):
        items[item_id].update(item.dict())
        return items[item_id]
    raise HTTPException(status_code=404, detail="Item não encontrado")

```

/

Rota DELETE /items/{item_id} (Remoção)

```python
@app.delete("/items/{item_id}")
async def delete_item(item_id: int):
    if 0 <= item_id < len(items):
        removed_item = items.pop(item_id)
        return removed_item
    raise HTTPException(status_code=404, detail="Item não encontrado")
```

/

Rota DELETE /items/{item_id} (Remoção)

```python
@app.delete("/items/{item_id}")
async def delete_item(item_id: int):
    if 0 <= item_id < len(items):
        removed_item = items.pop(item_id)
        return removed_item
    raise HTTPException(status_code=404, detail="Item não encontrado")
```

/

Vamos testar o POST

Agora no terminal, em vez de postman

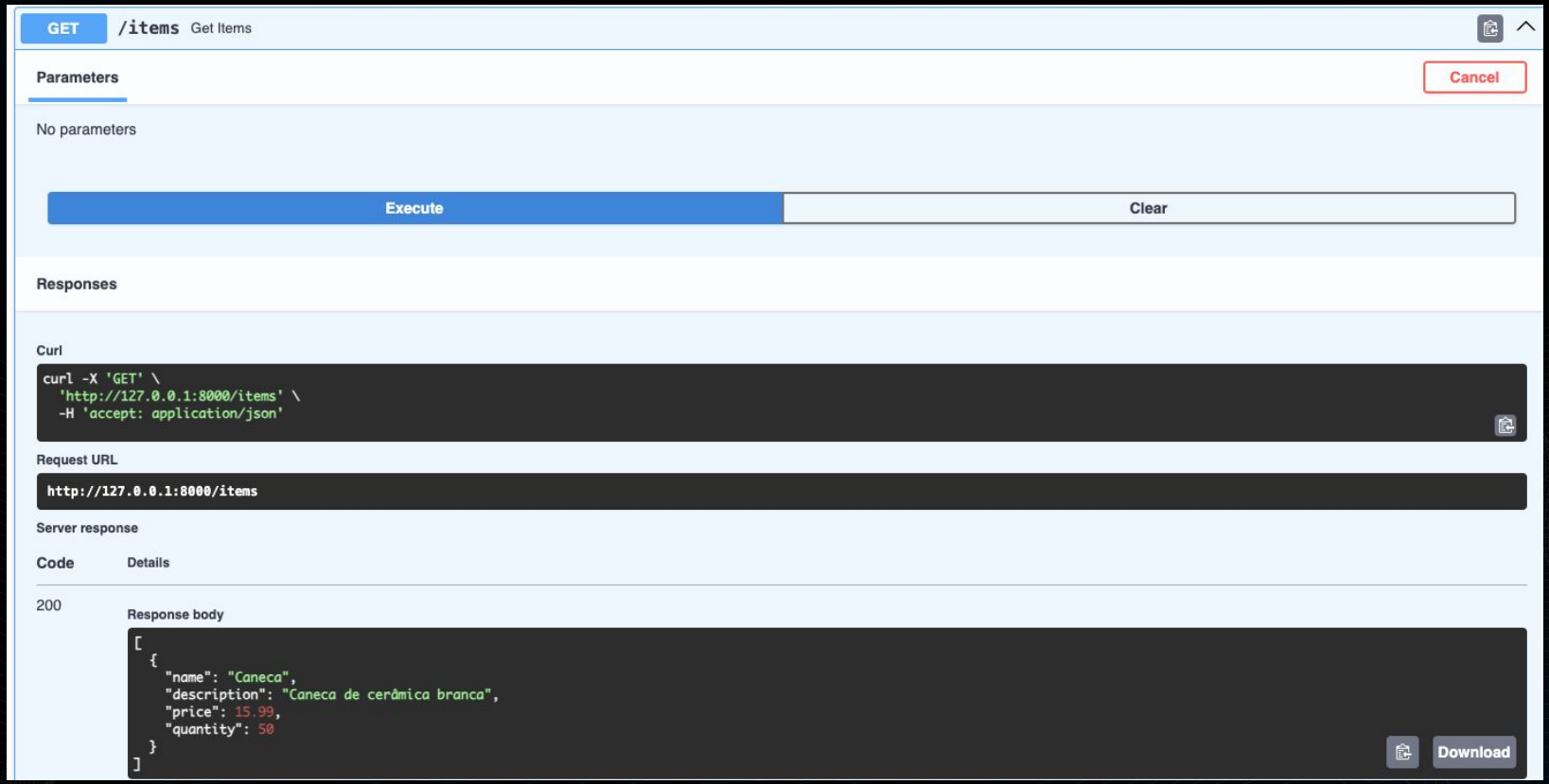
```
leonardopena@MacBook-Pro-de-Leonardo ~ % curl -X POST "http://127.0.0.1:8000/items" \
     -H "Content-Type: application/json" \
     -d '{
           "name": "Caneca",
           "description": "Caneca de cerâmica branca",
           "price": 15.99,
           "quantity": 50
         }'

{"name":"Caneca","description":"Caneca de cerâmica branca","price":15.99,"quantity":50}%
```

/

Vamos testar o POST

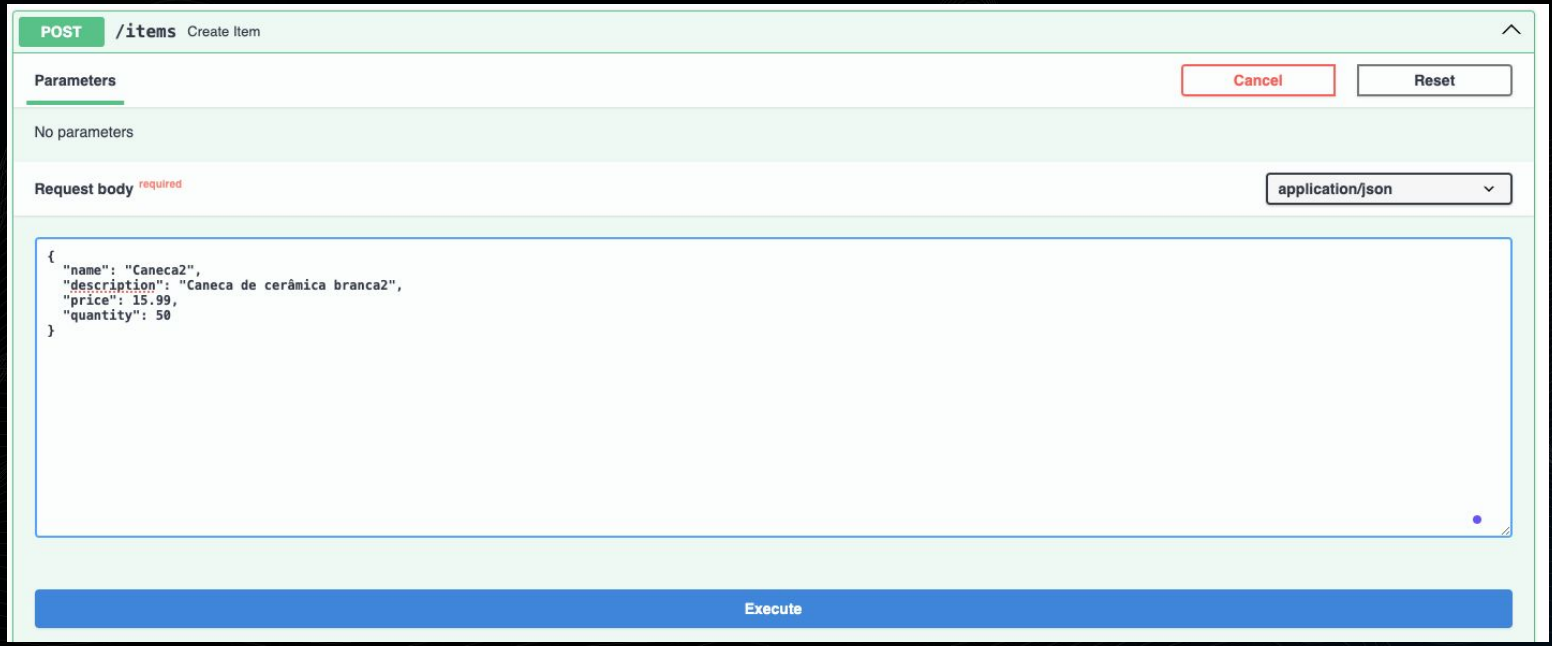
Agora no terminal, em vez de postman

```
leonardopena@MacBook-Pro-de-Leonardo ~ % curl -X POST "http://127.0.0.1:8000/items" \
     -H "Content-Type: application/json" \
     -d '{
           "name": "Caneca",
           "description": "Caneca de cerâmica branca",
           "price": 15.99,
           "quantity": 50
         }'

{"name":"Caneca","description":"Caneca de cerâmica branca","price":15.99,"quantity":50}%
```


GET
/items
Get Items
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/items' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/items
Server response
Code
Details
200
Response body
[
{
"name": "Caneca",
"description": "Caneca de cerâmica branca",
"price": 15.99,
"quantity": 50
}
]
Download

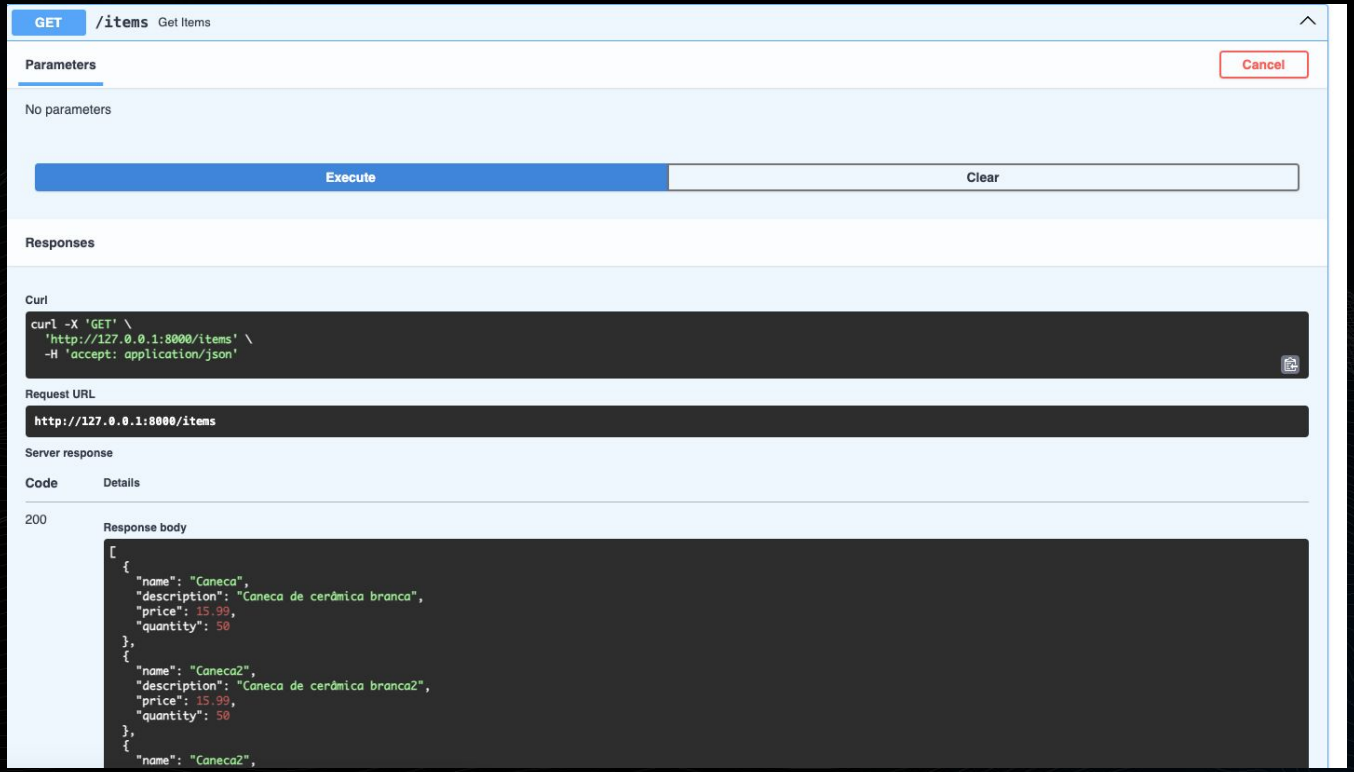
/

É possível testar direto no /docs também!



/

GET Atualizado



/

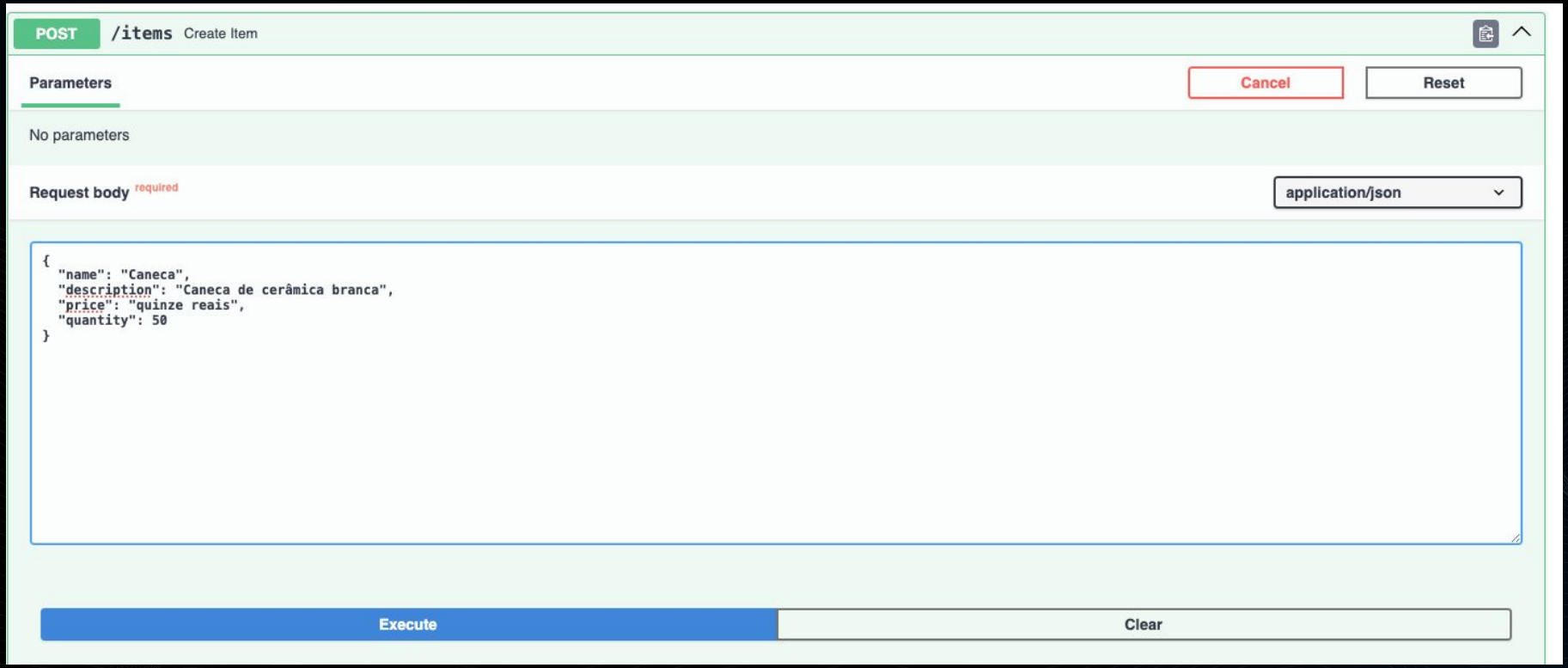
Lá atrás definimos a classe Item.
E se passarmos os argumentos errados?

```python
# Lista de itens em memória para operações CRUD
items = []  # armazena itens como dicionários

class Item(BaseModel):
    name: str             # nome do item
    description: str = None  # descrição opcional
    price: float = None   # preço opcional
    quantity: int = None  # quantidade opcional
```

/

Lá atrás definimos a classe Item.
E se passarmos os argumentos errados?

```python
# Lista de itens em memória para operações CRUD
items = []  # armazena itens como dicionários

class Item(BaseModel):
    name: str             # nome do item
    description: str = None  # descrição opcional
    price: float = None   # preço opcional
    quantity: int = None  # quantidade opcional
```


POST
/items
Create Item
Parameters
Cancel
Reset
No parameters
Request body
required
application/json
{
"name": "Caneca",
"description": "Caneca de cerâmica branca",
"price": "quinze reais",
"quantity": 50
}
Execute
Clear

/

Gera o erro

Code	Details
422	Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "type": "float_parsing",
      "loc": [
        "body",
        "price"
      ],
      "msg": "Input should be a valid number, unable to parse string as a number",
      "input": "quinze reais"
    }
  ]
}
```


Download

Response headers

/ Comparação Flask

FastAPI

Flask

- Em Flask, faríamos rotas com @app.route('/items', methods=['GET','POST']) etc.
- Precisaríamos converter request.json manualmente e retornar jsonify(...)
- Em FastAPI, tipagem e serialização são automáticas com Pydantic
- A doc e validação 422 são nativos

FIAP + alura