

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/DEL  
ETE



## 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



# FastAPI

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

# / Classe Item

```
from pydantic import BaseModel
```

```
40 # Lista de itens em memória para operações CRUD
41 items = [] # armazena itens como dicionários
42
43 class Item(BaseModel):
44     name: str # nome do item
45     description: str = None # descrição opcional
46     price: float = None # preço opcional
47     quantity: int = None # quantidade opcional
48
```

# /

## Rota GET /items (Leitura de Todos)

```
48  
49 @app.get("/items")  
50 async def get_items():  
51     return items  
52
```

### My FastAPI API 1.0.0 OAS 3.1

/openapi.json

API de Exemplo com FastAPI

Authorize



#### default



GET

/ Home



GET

/hello Hello



GET

/items Get Items





# Rota POST /items (Criação)

```
53 @app.post("/items", status_code=201)
54 ✓ async def create_item(item: Item):
55     items.append(item.dict())
56     return item
```

Vamos testar mais abaixo

The screenshot displays a REST client interface with a list of endpoints and a schemas section.

**Endpoints:**

- GET / Home
- GET /hello Hello
- GET /items Get Items
- POST /items Create Item

**Schemas:**

- HTTPValidationError > Expand all object
- Item > Expand all object
- ValidationError > Expand all object

# /

## Rota PUT

### /items/{item\_id}

### (Atualização)

```
58 @app.put("/items/{item_id}")
59 ∨ async def update_item(item_id: int, item: Item):
60 ∨     if 0 <= item_id < len(items):
61         items[item_id].update(item.dict())
62         return items[item_id]
63         raise HTTPException(status_code=404, detail="Item não encontrado")
64
```

# /

## Rota DELETE /items/{item\_id} (Remoção)

```
66 @app.delete("/items/{item_id}")
67 async def delete_item(item_id: int):
68     if 0 <= item_id < len(items):
69         removed_item = items.pop(item_id)
70         return removed_item
71     raise HTTPException(status_code=404, detail="Item não encontrado")
72
```

# /

## Rota DELETE /items/{item\_id} (Remoção)

```
66 @app.delete("/items/{item_id}")
67 async def delete_item(item_id: int):
68     if 0 <= item_id < len(items):
69         removed_item = items.pop(item_id)
70         return removed_item
71     raise HTTPException(status_code=404, detail="Item não encontrado")
72
```

# Vamos testar o POST

Agora no terminal,  
em vez de postman

```
Last login: Fri Dec 27 10:58:02 on cty3033
leonardopenda@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

```
Last login: Fri Dec 27 10:58:02 on cty3033
leonardopenda@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!

The screenshot shows a REST client interface with a light green header bar. The header contains a green tab labeled 'POST', the endpoint '/items', and the text 'Create Item'. On the right side of the header are 'Cancel' and 'Reset' buttons. Below the header, the 'Parameters' section is active, showing 'No parameters'. The 'Request body' section is also active, with a red 'required' label and a dropdown menu set to 'application/json'. The request body is a JSON object: { "name": "Caneca2", "description": "Caneca de cerâmica branca2", "price": 15.99, "quantity": 50 }. At the bottom, there is a large blue 'Execute' button.

**POST** /items Create Item

**Parameters** Cancel Reset

No parameters

**Request body** required application/json

```
{
  "name": "Caneca2",
  "description": "Caneca de cerâmica branca2",
  "price": 15.99,
  "quantity": 50
}
```

Execute

# /

# GET Atualizado

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
  },
  {
    "name": "Caneca2",
    "description": "Caneca de cerâmica branca2",
    "price": 15.99,
    "quantity": 50
  },
  {
    "name": "Caneca2",
```

/

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

```
# 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?

```
# 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

POSTECH

FIAP + alura