Laboratório Prático: Introdução a APIs Assíncronas com FastAPI e ASGI

1. Crie um arquivo chamado `main.py`.
Adicione o seguinte código:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root():
    return {"mensagem": "Olá, Sistemas Distribuídos!"}
```

Rode o servidor usando o Uvicorn e abra o navegador em `http://127.0.0.1:8000`.

```
uvicorn main:app --reload
```

2. O FastAPI usa a biblioteca Pydantic para validar dados automaticamente. Vamos criar uma estrutura para um **Item**.

No arquivo `main.py`, adicione/atualize:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()

# Modelo de Dados
class Item(BaseModel):
    id: int
    nome: str
    preco: float
    em_estoque: bool = True

banco_de_dados = [
    Item(id=1, nome="Caneta", preco=3.50),
    Item(id=2, nome="Caderno", preco=15.00),
    Item(id=3, nome="Mochila", preco=120.00)
]
```

3. Crie rotas para adicionar e listar itens. Note o uso de async def.

Adicione ao main.py:

```
# Rota para criar um item (POST)
@app.post("/itens/", response_model=Item)
async def criar_item(item: Item):
    for i in banco_de_dados:
        if i.id == item.id:
            raise HTTPException(status_code=400, detail="Item com este ID já existe")

    banco_de_dados.append(item)
    return item

# Rota para listar todos os itens (GET)
@app.get("/itens/", response_model=List[Item])
async def listar_itens():
    return banco_de_dados

# Rota para buscar um item específico (GET)
@app.get("/itens/{item_id}", response_model=Item)
async def ler_item(item_id: int):
    for item in banco_de_dados:
        if item.id == item_id:
            return item
    raise HTTPException(status_code=404, detail="Item não encontrado")
```

4. Com o servidor rodando, acesse: http://127.0.0.1:8000/docs.

Use o botão "Try it out" no endpoint **POST /itens/** para criar itens e depois **GET /itens/** para listá-los.

Tente criar um Item com dados inválidos intencionalmente. Peça para enviarem o campo preco (definido como float) como uma string de texto (ex: "trinta reais").

O que o Swagger retornou? Foi necessário escrever algum if no código para validar isso?

5. Para demonstrar o poder do assincronismo em sistemas distribuídos, vamos simular uma operação lenta (como consultar uma API externa ou um banco de dados pesado) que não bloqueia o servidor.

Importe a biblioteca asyncio no topo do arquivo.

Adicione esta nova rota especial:

```
@app.get("/processamento-pesado/")
async def processamento_pesado():
```

```
    # Simula uma espera de 5 segundos (ex: aguardando resposta de outro servidor)
    print("Iniciando processamento pesado...")
# O 'await' libera o servidor para atender outras requisições enquanto espera
    await asyncio.sleep(5)
    return {"status": "Processamento finalizado com sucesso"}
```

Teste a Concorrência:

- Abra duas abas do navegador.
- Na primeira, execute o endpoint /processamento-pesado/.
- Imediatamente vá na segunda aba e execute o /itens/.

Observação: Você notará que a lista de itens retorna *imediatamente*, mesmo com a outra aba ainda "carregando" os 5 segundos. Se fosse um servidor síncrono (WSGI) com 1 worker, a segunda aba ficaria travada esperando a primeira terminar.

6. Para validar o laboratório, implemente as seguintes funcionalidades no código acima:

a) Adicione uma rota **PUT** para atualizar o preço de um item existente.
b) Adicione uma rota **DELETE** para remover um item do banco de dados.

Teste as novas funcionalidades na documentação