CRUD - FastAPI

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Slides baseados no material do Professor Régis Magalhães

Agenda

- CRUD com FastAPI - persistindo em uma lista
- CRUD com FastAPI - persistindo em um csv

CRUD com FastAPI - persistindo em uma lista

```python
from typing import Union
from typing import List
from http import HTTPStatus
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    id: int
    nome: str
    valor: float
    is_oferta: Union[bool, None] = None

itens: List[Item] = []
```

Essa classe `Item` representa o modelo de dados do item, contendo:

- `id` (inteiro): Identificador do item.
- `nome` (string): Nome do item.
- `valor` (float): Valor do item.
- `is_oferta` (opcional, `bool`): Indica se o item está em promoção (`True` ou `False`) ou `None` se o valor não for fornecido.
-

CRUD com FastAPI - persistindo em uma lista

```python
@app.get("/")
def padrao():
    return {"msg": "Hello World"}

@app.get("/itens/{item_id}", response_model=Item)
def ler_item(item_id: int):
    for indice, item_atual in enumerate(itens):
        if item_atual.id == item_id:
            return item_atual
    raise HTTPException(status_code=HTTPStatus.NOT_FOUND,
detail="Item não encontrado.")
```

- **Primeira Rota** (`"/"`): Sempre retorna `{"msg": "Hello World"}`, útil para verificar se a API está ativa.
- **Segunda Rota** (`"/itens/{item_id}"`): Retorna um item específico com base no `item_id` ou lança um erro se o item não for encontrado.

CRUD com FastAPI - persistindo em uma lista

```python
@app.get("/itens/", response_model=List[Item])
def listar_itens():
    return itens

@app.post("/itens/", response_model=Item,
status_code=HTTPStatus.CREATED)
def adicionar_item(item: Item):
    if any(item_atual.id == item.id for item_atual in itens):
        raise HTTPException(status_code=400, detail="ID já
existe.")
    itens.append(item)
    return item
```

- **Terceira Rota (`"/itens/"`, método `GET`)**: Retorna todos os itens armazenados na lista `itens`, permitindo ao usuário visualizar todos os itens cadastrados.
- **Quarta Rota (`"/itens/"`, método `POST`)**: Adiciona um novo item à lista. Verifica se o `id` do item já existe para evitar duplicação. Caso o `id` já exista, retorna um erro `400 Bad Request`; caso contrário, adiciona o item e retorna `201 Created`.

CRUD com FastAPI - persistindo em uma lista

```python
@app.put("/itens/{item_id}", response_model=Item)
def atualizar_item(item_id: int, item_atualizado: Item):
    for indice, item_atual in enumerate(itens):
        if item_atual.id == item_id:
            if item_atualizado.id != item_id:
                item_atualizado.id = item_id
            itens[indice] = item_atualizado
            return item_atualizado
    raise HTTPException(status_code=HTTPStatus.NOT_FOUND,
detail="Item não encontrado.")
```

- **Quinta Rota (`"/itens/{item_id}"`, método `PUT`)**: Atualiza um item existente com base no `item_id`. Se o `id` fornecido para o item atualizado não corresponder ao `item_id`, ele é ajustado para o valor correto. Se o item não for encontrado, retorna um erro `404 Not Found`.

CRUD com FastAPI - persistindo em um csv

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import csv
import os

app = FastAPI()
CSV_FILE = "database.csv"

# Modelo de dados para o produto
class Produto(BaseModel):
    id: int
    nome: str
    preco: float
    quantidade: int
```

- `id` (inteiro): Identificador único do produto. É necessário para diferenciar cada produto individualmente.
- `nome` (string): Nome do produto, que descreve o que ele é.
- `preco` (float): Preço do produto, que indica o valor em moeda.
- `quantidade` (inteiro): Quantidade disponível do produto, usada para rastrear o estoque.

CRUD com FastAPI - persistindo em um csv

```
# Função para ler os dados do CSV
def ler_dados_csv():
    produtos = []
    if os.path.exists(CSV_FILE):
        with open(CSV_FILE, mode="r", newline="") as
file:
            reader = csv.DictReader(file)
            for row in reader:
                produtos.append(Produto(**row))
    return produtos
```

- Cria uma lista `produtos` para armazenar os dados.
- Verifica se o arquivo CSV existe.
- Abre o arquivo CSV para leitura.
- Lê cada linha como um dicionário e cria um objeto `Produto` com os dados.
- Adiciona cada `Produto` à lista `produtos`.
- Retorna a lista `produtos` com todos os produtos do CSV.

CRUD com FastAPI - persistindo em um csv

```
# Função para escrever os dados no CSV
def escrever_dados_csv(produtos):
    with open(CSV_FILE, mode="w", newline="") as
file:
        fieldnames = ["id", "nome", "preco",
"quantidade"]
        writer = csv.DictWriter(file,
fieldnames=fieldnames)
        writer.writeheader()
        for produto in produtos:
            writer.writerow(produto.dict())
```

- Abre o arquivo CSV em modo de escrita (`"w"`), criando ou substituindo o conteúdo existente.
- Define as colunas do CSV com `fieldnames`, correspondendo aos atributos de `Produto`.
- Inicializa um `DictWriter` para escrever dicionários no CSV.
- Escreve o cabeçalho do CSV com `writeheader()`.
- Para cada `produto` na lista `produtos`, converte-o para dicionário com `produto.dict()` e grava a linha no CSV.

CRUD com FastAPI - persistindo em um csv

```python
# Rota para obter todos os produtos
@app.get("/produtos", response_model=list[Produto])
def listar_produtos():
    return ler_dados_csv()

# Rota para obter um produto por ID
@app.get("/produtos/{produto_id}", response_model=Produto)
def obter_produto(produto_id: int):
    produtos = ler_dados_csv()
    for produto in produtos:
        if produto.id == produto_id:
            return produto
    raise HTTPException(status_code=404, detail="Produto não encontrado")
```

- **Primeira Rota (`"/produtos"`, método `GET`)**: Retorna todos os produtos cadastrados, lendo os dados diretamente do CSV.
- **Segunda Rota (`"/produtos/{produto_id}"`, método `GET`)**: Retorna um produto específico com base no `produto_id`. Se o produto não for encontrado, retorna um erro `404 Not Found`.

CRUD com FastAPI - persistindo em um csv

```
# Rota para criar um novo produto
@app.post("/produtos", response_model=Produto)
def criar_produto(produto: Produto):
    produtos = ler_dados_csv()
    if any(p.id == produto.id for p in produtos):
        raise HTTPException(status_code=400, detail="ID
já existe")
    produtos.append(produto)
    escrever_dados_csv(produtos)
    return produto
```

- **Terceira Rota (`"/produtos"`, método `POST`)**: Cria um novo produto. Verifica se o `id` do produto já existe; se existir, retorna um erro `400 Bad Request`. Caso contrário, adiciona o produto à lista, salva no CSV e retorna o produto criado.

CRUD com FastAPI - persistindo em um csv

```python
# Rota para atualizar um produto
@app.put("/produtos/{produto_id}", response_model=Produto)
def atualizar_produto(produto_id: int, produto_atualizado:
Produto):
    produtos = ler_dados_csv()
    for i, produto in enumerate(produtos):
        if produto.id == produto_id:
            produtos[i] = produto_atualizado
            escrever_dados_csv(produtos)
            return produto_atualizado
    raise HTTPException(status_code=404, detail="Produto
não encontrado")
```

- **Quarta Rota (`"/produtos/{produto_id}"`, método `PUT`)**: Atualiza um produto existente com base no `produto_id`. Se o produto for encontrado, substitui seus dados pelos do `produto_atualizado`, salva no CSV e retorna o produto atualizado. Se o produto não for encontrado, retorna um erro `404 Not Found`.

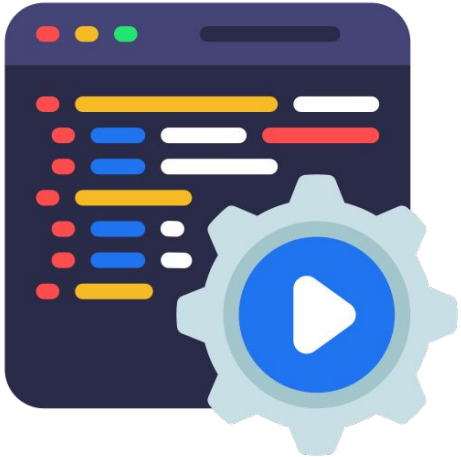
CRUD com FastAPI - persistindo em um csv

```python
# Rota para deletar um produto
@app.delete("/produtos/{produto_id}", response_model=dict)
def deletar_produto(produto_id: int):
    produtos = ler_dados_csv()
    produtos_filtrados = [produto for produto in produtos
if produto.id != produto_id]
    if len(produtos) == len(produtos_filtrados):
        raise HTTPException(status_code=404,
detail="Produto não encontrado")
    escrever_dados_csv(produtos_filtrados)
    return {"mensagem": "Produto deletado com sucesso"}
```

- **Quinta Rota (`"/produtos/{produto_id}"`, método `DELETE`)**: Deleta um produto com base no `produto_id`. Filtra a lista para excluir o produto com o `id` correspondente. Se o produto não for encontrado, retorna um erro `404 Not Found`. Caso contrário, salva a lista atualizada no CSV e retorna uma mensagem de sucesso: `"Produto deletado com sucesso"`.

Referências

- https://fastapi.tiangolo.com/
- https://fastapi.tiangolo.com/tutorial/



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br