Universidade Federal do Ceará
Campus Quixadá
QXD0099 - Desenvolvimento de Software para Persistência
Prof. Francisco Victor da Silva Pinheiro

Trabalho Prático 1 - Desenvolvimento de uma API REST para Gerenciamento de Entidades com Persistência em CSV com FastAPI

Descrição Geral do Trabalho

Neste trabalho, você irá desenvolver uma aplicação web simples que utiliza uma API REST criada com FastAPI para gerenciar entidades relacionadas ao domínio escolhido no Trabalho Prático 1 (TP1). O objetivo é praticar o desenvolvimento de APIs e manipulação de dados em Python, utilizando arquivos CSV para armazenamento dos dados. Além disso, funcionalidades extras como compactação de arquivos, cálculo de hash e logging (registro de operações) serão implementadas para simular um cenário real de aplicação com maior nível de controle e usabilidade.

Passo a Passo do Trabalho

1. Definir uma entidade e criar uma classe python

- **Objetivo:** Escolher uma entidade (um "objeto" ou "coisa" representativa) que faça sentido no contexto do domínio escolhido no TP1. Essa entidade deve ter **pelo menos cinco atributos**.
- **Exemplo:** Se o domínio for "vendas", vocês poderiam definir uma entidade chamada **Produto** com atributos como **id, nome, categoria, preco** e **data_criacao**.
- **Implementação:** Você deve implementar uma classe Python para representar essa entidade usando a biblioteca **pydantic**, que ajudará a garantir que os dados fornecidos estão no formato esperado.

2. Criar uma API REST com FastAPI

Usando FastAPI, vocês vão criar endpoints para implementar cada funcionalidade solicitada. Cada funcionalidade será implementada em um endpoint específico. Abaixo estão os detalhes de cada funcionalidade que a API deverá oferecer.

Funcionalidades da API

F1. Inserir uma entidade no arquivo CSV

- **Objetivo:** Implementar um endpoint para **cadastrar** uma nova entidade no sistema.
- **Detalhes:** Quando o endpoint for acessado com um JSON contendo os dados da entidade, a API deverá adicionar essa entidade ao arquivo CSV, usando o modo de

"append" (ou seja, adicionando ao final do arquivo sem substituir os dados existentes).
- **Exemplo:** Enviar um JSON com os dados de um novo **Produto** e salvar esses dados no CSV, cada linha representando um produto diferente.

F2. Listar todas as entidades do CSV

- **Objetivo:** Implementar um endpoint para **retornar todos os registros** da entidade.
- **Detalhes:** O endpoint deverá ler o arquivo CSV e retornar todas as entidades cadastradas em formato JSON. Isso permitirá visualizar todos os dados que foram cadastrados até o momento.
- **Exemplo:** Se houver três produtos cadastrados no CSV, ao acessar esse endpoint, o usuário verá um JSON contendo os três produtos.

F3. CRUD completo da entidade

- **Objetivo:** Implementar endpoints para as operações de **Create (criar), Read (ler), Update (atualizar) e Delete (excluir)** para a entidade.
- **Detalhes:** O CRUD permite que o usuário realize todas as operações fundamentais para gerenciar uma entidade. Em todos os casos, o arquivo CSV deverá ser atualizado conforme as mudanças. As operações incluem:
    - **Criar:** Já implementado no F1.
    - **Ler:** Listagem de todos os registros, conforme F2.
    - **Atualizar:** Modificar um registro específico no CSV.
    - **Excluir:** Remover um registro específico do CSV.
- **Exemplo:** O usuário poderá modificar o nome de um produto ou excluir um produto específico.

F4. Mostrar a quantidade de entidades

- **Objetivo:** Implementar um endpoint para **mostrar a quantidade total** de entidades cadastradas.
- **Detalhes:** Este endpoint deve contar o número de linhas no arquivo CSV e retornar essa contagem ao usuário. Importante: deve refletir a contagem real, mesmo que alguém edite o CSV externamente (fora do sistema).
- **Exemplo:** Se há 10 produtos cadastrados no CSV, o endpoint deve retornar **{ "quantidade": 10 }**.

F5. Compactar o arquivo CSV em ZIP

- **Objetivo:** Implementar um endpoint para **compactar o arquivo CSV em um arquivo ZIP** e disponibilizar o download do arquivo compactado.
- **Detalhes:** Esse endpoint deve criar um arquivo ZIP contendo o CSV e permitir que o usuário faça o download. Isso é útil para backup ou exportação dos dados.
- **Exemplo:** Quando acessado, o endpoint deve retornar o arquivo **produtos.zip** contendo **produtos.csv**.

F6. Filtrar entidades por atributos específicos

- **Objetivo:** Implementar um endpoint para **filtrar as entidades cadastradas** com base em atributos específicos.
- **Detalhes:** O usuário pode fornecer parâmetros de filtragem, como categoria ou intervalo de preço. A API deve retornar apenas às entidades que atenderem aos critérios definidos.
- **Exemplo:** Se a entidade **Produto** tiver um atributo **categoria**, o usuário poderá filtrar para ver apenas os produtos de uma categoria específica, como "Eletrônicos".

F7. Retornar o Hash SHA256 do Arquivo CSV

- **Objetivo:** Implementar um endpoint para **calcular e retornar o hash SHA256 do arquivo CSV**.
- **Detalhes:** Esse endpoint deve calcular o hash SHA256 do arquivo CSV e retornar o valor do hash em formato de texto. Esse hash pode ser utilizado para verificar a integridade dos dados, comparando se a versão do CSV não foi alterada desde a última vez que o hash foi gerado.
- **Exemplo:** Quando acessado, o endpoint deve retornar algo como **{ "hash_sha256": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" }**.

Funcionalidade Bônus

F8. Implementar um sistema de logs (0,5 pontos extras)

- **Objetivo:** Adicionar um sistema de **logging** para registrar as operações realizadas na API.
- **Detalhes:** Cada operação executada na API, como inserções, exclusões ou falhas, deve ser registrada em um arquivo de log para fins de auditoria e monitoramento. Esse log ajudará a rastrear o histórico de interações com a API.
- **Exemplo:** Quando um novo produto for adicionado, o sistema deve registrar uma mensagem como "Produto inserido com sucesso" com a data e hora da operação.

F9. Implementar um front-end (1,5 pontos extras)

- **Objetivo:** Desenvolver uma interface gráfica para que os usuários possam interagir com a API de forma visual e intuitiva.
- **Detalhes:** A interface deve permitir que os usuários realizem operações principais, como inserir, consultar, atualizar e excluir dados, diretamente através do front-end. O layout deve ser intuitivo, responsivo e facilitar o uso das funcionalidades da API.
- **Exemplo:** Quando o usuário adicionar um novo produto por meio do front-end, a interface deve mostrar uma mensagem de confirmação como "Produto inserido com sucesso" e atualizar a lista de produtos exibida em tempo real.

"A melhor maneira de predizer o futuro é inventá-lo."
Alan Kay