API REST de Personagens de Anime — Flask + Swagger

Este projeto é uma implementação de uma API RESTful para gerenciar uma coleção de personagens de anime, desenvolvida como parte de uma atividade acadêmica sobre a arquitetura cliente-servidor e o uso de HTTP.

O projeto inclui:

- Uma API construída com Python, Flask e Flask-RESTX.
- Um cliente de console não interativo para demonstrar o consumo da API.
- Um cliente de console interativo para manipulação manual dos dados.
- Documentação automática da API via Swagger UI.
- Coleções e arquivos de requisição para ferramentas populares de teste de API (Postman, Insomnia, VS Code REST Client).

> Para evidências detalhadas (cURL, Postman, Insomnia, requests.http e checklist), veja o documento **EVIDENCIAS.md** neste mesmo projeto.

1) Estrutura do Projeto

```
/02_Cliente-servidor simples_Python
├── evidencias_imagens/     # Pasta para imagens do EVIDENCIAS.md
├── api.py                  # O código do servidor da API (Flask-RESTX)
├── cliente.py              # Cliente de console não interativo (script)
├── cliente_interativo.py   # Cliente de console interativo (CLI)
├── requirements.txt        # Dependências do Python
├── requests.http           # Requisições para a extensão REST Client (VS Code)
├── postman_collection.json # Coleção para o Postman
├── insomnia_export.json    # Exportação de dados para o Insomnia
└── README.md               # Este arquivo
```

2) Features

- **Operações CRUD completas**: Crie, leia, atualize e delete personagens.
- **Filtragem e Paginação**: A listagem de personagens suporta filtros por `universo` e paginação com `limit` e `offset`.
- **Documentação Interativa**: A interface do Swagger UI, gerada automaticamente, permite explorar e testar os endpoints da API diretamente pelo navegador.
- **Validação de Dados**: Os dados enviados para a API são validados para garantir a consistência.
- **Múltiplas Formas de Consumo**: O projeto oferece diversos exemplos de como consumir a API, desde clientes de console até ferramentas especializadas.

3) Tecnologias Utilizadas

- **Backend**:
  - **Python 3**: Linguagem de programação principal.
  - **Flask**: Micro-framework web para a criação do servidor.
  - **Flask-RESTX**: Extensão do Flask que facilita a criação de APIs RESTful e a geração de documentação Swagger.
- **Cliente**:
  - **Python 3**: Para os clientes de console.
  - **Requests**: Biblioteca para realizar requisições HTTP.
- **Ferramentas de Teste**:
  - **Postman**: Plataforma para teste de APIs.
  - **REST Client**: Extensão do Visual Studio Code para realizar requisições HTTP a partir de um arquivo de texto.

4) Como rodar a API

> Siga os passos abaixo para configurar e executar o projeto em seu ambiente local.

Pré-requisitos

- Python 3.8 ou superior
- pip (gerenciador de pacotes do Python)

1. Clone o Repositório:

```
# (Exemplo, caso estivesse em um repositório git)
git clone https://github.com/seu-usuario/seu-repositorio.git
cd seu-repositorio
```

2. Crie e Ative um Ambiente Virtual (recomendado):

```
python -m venv .venv
source .venv/bin/active
# Windows: .venv\Scripts\activate
# macOS/Linux: source .venv/bin/activate
```

3. Instale as Dependências:
Com o ambiente virtual ativado, instale as bibliotecas necessárias:

```
pip install -r requirements.txt
# ou, se preferir:
# pip install flask flask-restx
```

4. Execute a API:
Para iniciar o servidor da API, execute o seguinte comando:

```
python api.py
```

5. Acesse a Documentação Swagger

Com a API em execução, abra seu navegador e acesse a Base URL para ver a documentação interativa do Swagger UI.

- Base URL: **http://127.0.0.1:5000**
- Swagger UI: **http://127.0.0.1:5000/** (interface interativa)

6) Endpoints

Método	Caminho	Descrição	Obs
GET	`/personagens/`	Lista personagens	Filtros: `universo`, `limit`, `offset`
POST	`/personagens/`	Cria personagem	Body JSON obrigatório
GET	`/personagens/{id}`	Obtém por ID	—
PUT	`/personagens/{id}`	Atualiza personagem	Substitui os campos obrigatórios
DELETE	`/personagens/{id}`	Remove personagem	—
GET	`/`	Saúde/raiz	Link para Swagger UI

Modelo (body de entrada):

```
{
  "nome": "Naruto Uzumaki",
  "universo": "Naruto",
  "poder_principal": "Rasengan"
}
```

Códigos de status usados:

- 200 (OK);
- 201 (Criado);
- 204 (Sem conteúdo);
- 400 (Inválido), 404 (Não encontrado).

7) Cliente Console

Para consumir a API através dos clientes de console, abra um novo terminal (mantendo a API em execução) e execute um dos seguintes comandos:

Cliente não interativo:

```
python cliente.py
```

Cliente interativo:

```
python cliente_interativo.py
```

Observação: Os clientes estão configurados para se conectar à API em http://127.0.0.1:5000. Se você alterar a porta da API, precisará ajustar a variável de ambiente API_BASE_URL ao executar os clientes. Por exemplo:

```
BASE_URL = "http://127.0.0.1:5000/personagens"
# ou via variável de ambiente:
# API_BASE_URL=http://127.0.0.1:5000/personagens python cliente.py
```