

## Guia do Usuário da API de Personagens

### Introdução:

Bem-vindo ao Manual do Usuário da API de Gerenciamento de Personagens, esse documento foi elaborado com o intuito de orientar ao funcionamento das funcionalidades na API de Personagens. Você irá encontrar nesse documento informações sobre todas as funcionalidades (adicionar, consultar, atualizar e deletar personagens).

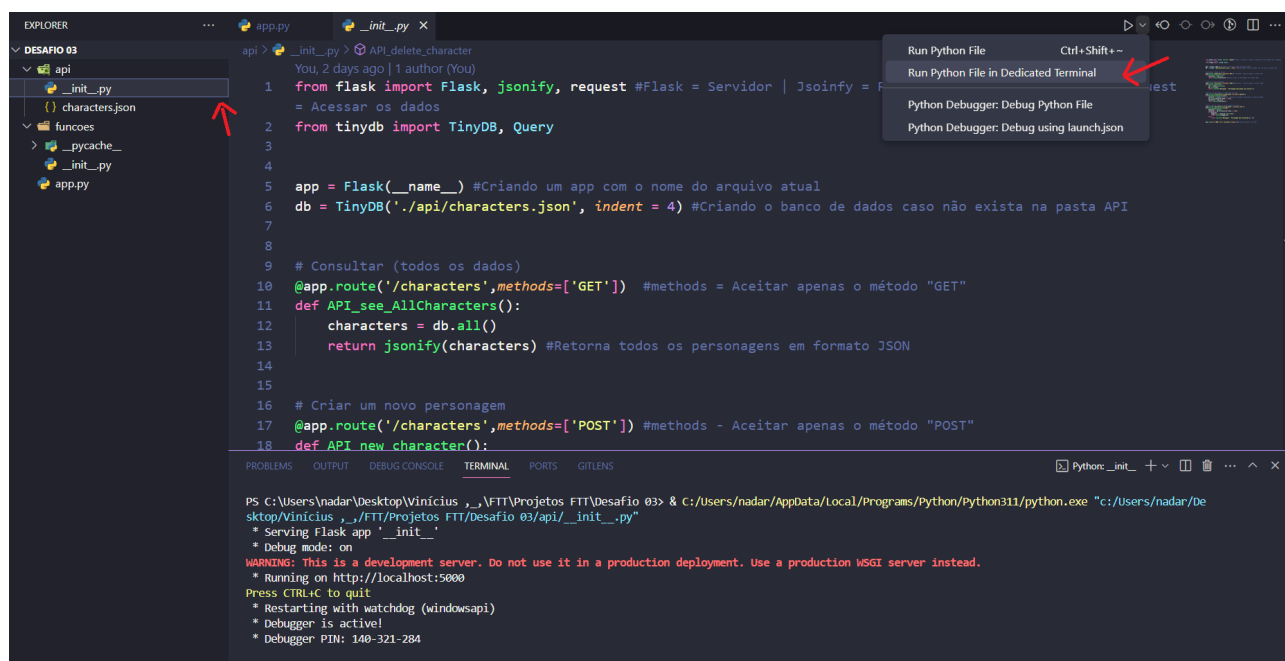
### Acesso a API:

Você deve executar o programa da API antes de executar o Gerenciador de Personagens, já que, como as interações do programa principal são feitas com a API, caso o programa não esteja executado ele irá dar um erro, pois tentará fazer alguma requisição (utilizando os métodos GET e POST ou DELETE) e não irá conseguir.

Siga esse caminho:

*api* → *\_\_init\_\_.py*

Execute esse programa em um terminal dedicado, veja:



The screenshot shows a code editor with a file explorer on the left. The file explorer shows a project named 'DESAFIO 03' with a subdirectory 'api' containing files 'characters.json', '\_\_init\_\_.py', 'funcoes', '\_\_pycache\_\_', and 'app.py'. The 'api' directory is selected, and the 'characters.json' file is highlighted. The main editor shows the content of 'api/\_\_init\_\_.py'. The code defines a Flask application with routes for getting all characters and creating a new character. The terminal at the bottom shows the command to run the application: 'C:\Users\nadar\Desktop\Vinicius > .\FTT\Projetos FTT\Desafio 03> & C:/Users/nadar/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/nadar/Desktop/Vinicius > .\FTT\Projetos FTT\Desafio 03/api/\_init\_.py"'. The output shows the application running on http://localhost:5000.

```
1 from flask import Flask, jsonify, request #Flask = Servidor | jsonify = F
2 = Acessar os dados
3 from tinydb import TinyDB, Query
4
5 app = Flask(__name__) #Criando um app com o nome do arquivo atual
6 db = TinyDB('./api/characters.json', indent = 4) #Criando o banco de dados caso não exista na pasta API
7
8
9 # Consultar (todos os dados)
10 @app.route('/characters', methods=['GET']) #methods = Aceitar apenas o método "GET"
11 def API_see_AllCharacters():
12     characters = db.all()
13     return jsonify(characters) #Retorna todos os personagens em formato JSON
14
15
16 # Criar um novo personagem
17 @app.route('/characters', methods=['POST']) #methods - Aceitar apenas o método "POST"
18 def API_new_character():
```

```
PS C:\Users\nadar\Desktop\Vinicius > .\FTT\Projetos FTT\Desafio 03> & C:/Users/nadar/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/nadar/Desktop/Vinicius > .\FTT\Projetos FTT\Desafio 03/api/_init_.py"
* Serving Flask app '_init_'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 140-321-284
```

Após seguir todos os passos citados anteriormente, o servidor estará disponível na URL de servidor local. Este é o servidor que você pode usar para desenvolvimento e testes.

URL da API Local: <http://localhost:5000>

Link para a documentação OpenAPI(Swagger): <https://app.swaggerhub.com/apis/ViniciusFlores/api-characters/1.0.0#>

## Métodos da API:

### Consultar todos os personagens

- **Método:** GET
- **Endpoint:** `‘/characters’`
- **Descrição:** Este método retorna a lista de todos os personagens disponíveis na base de dados.
- **Parâmetros:** Nenhum.
- **Resposta de Sucesso:** Retorna uma lista de personagens no formato JSON.

### Adicionar um novo personagem

- **Método:** POST
- **Endpoint:** `‘/characters’`
- **Descrição:** Este método permite que você adicione um novo personagem ao banco de dados da API.
- **Parâmetros:** Nenhum.
- **Resposta de Sucesso:** Retorna uma mensagem de confirmação.

### Conseguir um personagem pelo nome

- **Método:** GET
- **Endpoint:** `‘/characters/{name}’`
- **Descrição:** Este método permite você recuperar as informações de um personagem específico.
- **Parâmetros:** Nome do personagem para consulta detalhada.
- **Resposta de Sucesso:** Retorna todas as informações do personagem em formato JSON.
- **Resposta de Erro:** Retorna uma mensagem de erro se o personagem não existe no banco de dados ou ocorreu algum problema.

### Deletar um personagem pelo nome

- **Método:** DELETE
- **Endpoint:** `‘/characters/{name}’`
- **Descrição:** Este método permite você deletar um personagem já cadastrado no banco de dados.
- **Parâmetros:** Nome do personagem para remoção.
- **Resposta de Sucesso:** Retorna uma mensagem de sucesso.
- **Resposta de Erro:** Retorna uma mensagem de erro se o personagem não existe ou ocorreu algum problema.

### Atualizar informações de um personagem pelo nome

- **Método:** PUT
- **Endpoint:** `‘/characters/{name}’`
- **Descrição:** Este método permite você atualizar informações de um personagem já cadastrado no banco de dados.
- **Parâmetros:** Nome do personagem para atualização.
- **Resposta de Sucesso:** Retorna uma mensagem de sucesso.
- **Resposta de Erro:** Retorna uma mensagem de erro se o personagem não existe ou ocorreu algum problema.

## Estrutura de dados:

A API utiliza o seguinte formato para personagens:

- **Nome:** Nome do personagem.
- **Descrição:** Descrição do personagem.
- **Link:** URL da imagem do personagem, a imagem irá abrir no navegador padrão no usuário (Todas as instruções para como conseguir o link da imagem está disponível no programa).
- **Programa:** Programa do animador.
- **Animador:** Nome do animador do personagem.

## Conclusão:

Espero que esse Manual do Usuário da API de Gerenciamento de Personagens seja útil para você enquanto usa a minha API, obrigado pela atenção!