DESENVOLVIMENTO DE APIs E MICROSSERVIÇOS

6

Uso da biblioteca Requests, avançado

Lucas Mendes Marques Gonçalves

Resumo

É possível baixar informações das mais diversas na internet. Mas nos interessa integrar essas informações em aplicações. Para isso, temos que automatizar o processo de acesso. Nesse texto, vamos nos aprofundar nesse processo, usando os verbos HTTP e os códigos de status.

6.1. Requests, verbo POST

Quando desejamos enviar arquivos para o servidor, podemos usar os verbos PUT ou POST. Na biblioteca requests, isso é muito simples, como podemos ver abaixo:

Codificação 6.1. Requests, verbo post

```
import requests

def adiciona_corredor(nome,tempo, id):

   url = "http://localhost:5000/corredores"

   dici_corredor = {"nome":nome, "tempo": tempo, "id": id}

   r = requests.post(url, json=dici_corredor)

   return True
```

Fonte: do autor, 2021

No exemplo, montamos um dicionário dici_corredor, `dici_corredor = {"nome":nome, "tempo": tempo, "id": id}` e o enviamos, `r = requests.post(url, json=dici_corredor)` convertendo (automaticamente) para o formato JSON (um txt com formatação específica para enviar e receber dicionários e listas).

Observe como foi fácil escolher o verbo http POST, na linha `r = requests.post(url, json=dici_corredor)`

6.2. Requests, verbo DELETE, código de status

Codificação 6.2. Requests, verbo delete

```
import requests
def deleta_mais_lento():
    url = "http://localhost:5000/corredores/maior_tempo"
    r = requests.delete(url)
    if r.status_code == 500:
        return "nao é possivel remover de uma lista vazia"
    return "ok"
```

Fonte: do autor, 2021

Observe como é simples realizar um pedido http com o verbo delete: `r = requests.delete(url)` e também como é fácil detectar qual foi o resultado no nosso pedido usando o código de status retornado pelo servidor: `if r.status_code == 500:`

6.3. Preparação do computador para os exercícios

Na aula em vídeo, resolvemos uma sequência de exercícios usando a biblioteca requests. Essa sequência executa conectando em um servidor de corridas, que marca os tempos de corredores famosos.

Para rodar o servidor e poder conectar ele com seus exercícios, você deve fazer o seguinte:

Abrir o cmd do windows e rodar os seguintes comandos:

1. pip install --user flask
2. pip install --user requests

Depois disso, poderá executar o servidor fornecido usando o comando:

python corredores_server.py (mas antes terá que abrir no cmd, o diretório onde baixou o programa corredores_server.py).

Se você tiver problemas com esse processo, existem instruções de debug no fim do arquivo. Se estiver usando MAC ou GNU/linux, substitua o comando pip pelo comando pip3.

6.4. Exercícios

Abaixo seguem os enunciados dos exercícios. Pratique, interagindo com o servidor via Postman para entender o servidor.

As soluções seguem no próximo tópico (mas, provavelmente, vai ser mais interessante ver as soluções no vídeo).

Codificação 6.3. Exercício

```
import requests


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores do servidor de corredoes
# via GET, e devolve uma lista de todos os corredores



# faça uma função usando a biblioteca requests
# que acessa a URL /corredores do servidor de corredoes
# via POST, enviando um dicionário de um novo corredor.
# Um corredor tem os campos "nome", "tempo" e "id"



# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/maior_tempo do servidor
# de corredoes
# via GET, e retorne o nome do corredor
# mais lento

# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/maior_tempo do servidor
# de corredoes
# via DELETE, causando a remoção dos dados do corredor
# mais lento.
# Infelizmente o servidor tem um bug no caso em que a lista
# de corredores está vazia, mas vamos tratar esse bug
# no nosso cliente

# porque a funcionalidade anterior consiste em um erro
# de design no servidor corredores?
```

```
# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via GET,
# sua função deve retornar o nome do corredor em questão
# o o seu melhor tempo, em uma tupla


#como eu trataria o erro 404 e informaria o meu usuário?

# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via DELETE,
# causando a remoção dos dados do corredor
# mais lento


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via PUT,
# e você deverá enviar um dicionário com o novo tempo,
# para atualizar
# o tempo atual do corredor. O tempo deverá ser menor
# do que o tempo atual, caso contrário, o servidor
# lançará um erro, que você deve tratar
```

Fonte: do autor, 2021

6.5. Resolução dos exercícios

Segue abaixo, o gabarito dos exercícios:

Codificação 6.4. Resolução do exercício

```
import requests


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores do servidor de corredoes
# via GET, e devolve uma lista de todos os corredores


def todos_corredores():
    url = "http://localhost:5000/corredores"
    #conectar na URL usando o verbo GET?
    r = requests.get(url)
    lista = r.json()
    return lista


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores do servidor de corredoes
# via POST, enviando um dicionário de um novo corredor.
# Um corredor tem os campos "nome", "tempo" e "id"


def adiciona_corredor(nome,tempo, id):
    url = "http://localhost:5000/corredores"
    dici_corredor = {"nome":nome, "tempo": tempo, "id": id}
    r = requests.post(url, json=dici_corredor)
    return True


# faça uma função usando a biblioteca requests
```

```
# que acessa a URL /corredores/maior_tempo do servidor
# de corredoes
# via GET, e retorne o nome do corredor
# mais lento
def mais_lento():
    url = "http://localhost:5000/corredores/maior_tempo"
    r = requests.get(url)
    #voltou um dicionário. Como eu leio mesmo?
    dici_corredor = r.json()
    return dici_corredor['nome']


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/maior_tempo do servidor
# de corredoes
# via DELETE, causando a remoção dos dados do corredor
# mais lento.
# Infelizmente o servidor tem um bug no caso em que a lista
# de corredores está vazia, mas vamos tratar esse bug
# no nosso cliente
def deleta_mais_lento():
    url = "http://localhost:5000/corredores/maior_tempo"
    r = requests.delete(url)
    if r.status_code == 500:
        return "nao é possivel remover de uma lista vazia"
    return "ok"


# porque a funcionalidade anterior consiste em um erro
# de design no servidor corredores?

'''
```

```
O verbo DELETE deveria ser IDEMPOTENTE.
Deveria ser o caso que a segunda chamada não causa novo efeito colateral
'''

# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via GET,
# sua função deve retornar o nome do corredor em questão
# o o seu melhor tempo, em uma tupla

def corredor_por_id(id):
    url = f"http://localhost:5000/corredores/{id}"
    r = requests.get(url)
    dic_retornado = r.json()
    if r.status_code == 404:
        return "corredor nao existe"
    nome = (dic_retornado['corredor'])['nome']
    tempo = dic_retornado['corredor']['tempo']

    return (nome, tempo)

#como eu trataria o erro 404 e informaria o meu usuário?

# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via DELETE,
# causando a remoção dos dados do corredor
# mais lento
```

```
def deletar_por_id(id):
    url = f"http://localhost:5000/corredores/{id}"
    r = requests.delete(url)
    if r.status_code == 404:
        return "corredor nao existe"
    return "ok"


# faça uma função usando a biblioteca requests
# que acessa a URL /corredores/ID do servidor
# de corredores, onde ID é um código numérico.
# O acesso ocorrerá via PUT,
# e você deverá enviar um dicionário com o novo tempo,
# para atualizar
# o tempo atual do corredor. O tempo deverá ser menor
# do que o tempo atual, caso contrário, o servidor
# lançará um erro, que você deve tratar


def novo_tempo(id, tempo_enviado):
    url = f"http://localhost:5000/corredores/{id}"
    r = requests.put(url, json={"tempo": tempo_enviado})
    if r.status_code == 400:
        return "tempo nao atualizado por ser maior do que o record"
    if r.status_code == 404:
        return "corredor nao encontrado"
    return "ok"
```

Fonte: do autor, 2021

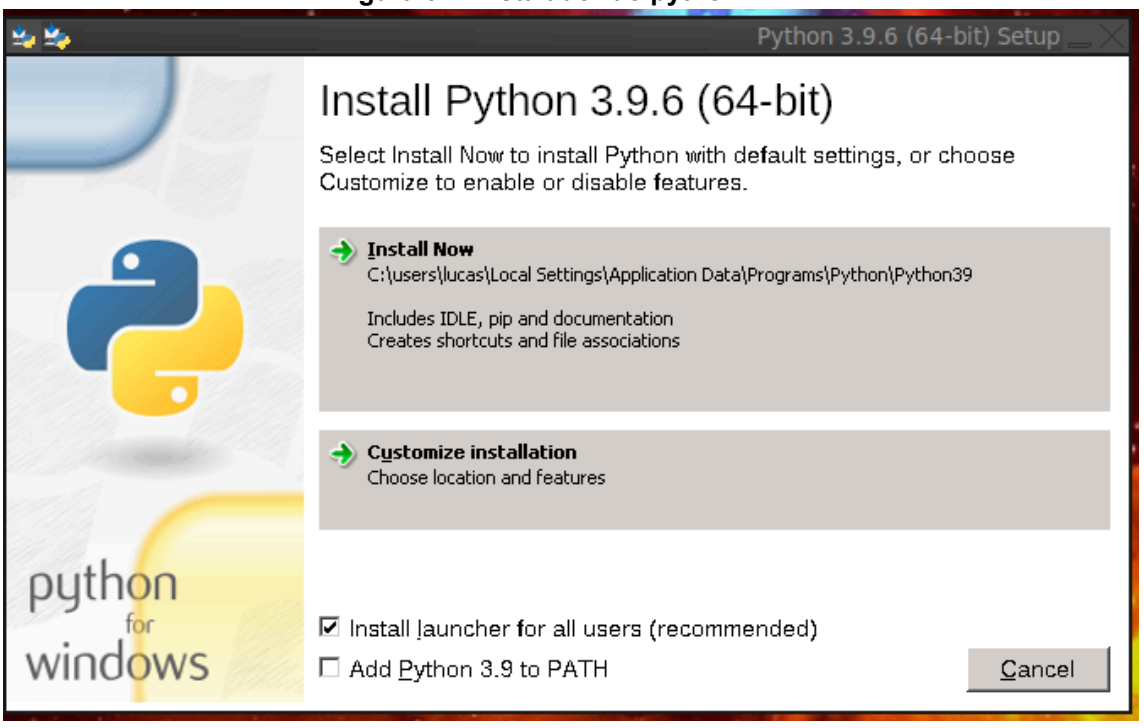
6.6. Solução de problemas

Se houver o erro: **O comando pip não é um programa válido - 'pip' não é reconhecido como um comando interno ou externo.** Siga os passos abaixo para resolver o problema:

1) Se você está usando linux ou mac, rode o comando de instalação usando pip3 no lugar do pip;
2) Se você está usando windows, experimente o comando python no cmd. Se funcionar (ou seja, o python funciona e o pip não), sua situação não é usual. Peça ajuda via classroom;
3) Se ambos os comandos (pip e python) não funcionarem no cmd, reinstalar o python deve resolver.

Ao reinstalar, marcar a opção "adicionar o python no path" ou "adicionar o python nas variáveis de ambiente". Isso faz com que os comandos "python" e "pip" passem a ser comandos válidos no cmd.

Figura 6.1. Instalador do python



Fonte: Python, 2021

Na figura 6.1, vemos que a caixa **Add Python 3.9 to PATH**, está desmarcada. Encontre essa opção na parte debaixo da imagem. A mesma faz com que os comandos **python** e **pip** estejam disponíveis no **cmd**. Por isso, deve ser marcada.

Depois de desinstalar e reinstalar, feche o cmd e abra um novo, para ele carregar os novos comandos.

Referências

Múltiplos autores. **Requests: HTTP para humanos.** Disponível em https://docs.python-requests.org/pt_BR/latest/. Acesso em 22 ago. 2021. (versão utilizável, mas desatualizada. Em português)

Múltiplos autores. **Requests: HTTP for humans.** Disponível em https://docs.python-requests.org/en/master/. Acesso em 22 ago. 2021.