BOSCH
Invented for life

Aula 3.7 - HTTP + Python

inside.Docupedia Export

Author: Lundgren Daniel (CtP/ETS)
Date: 15-Dec-2021 15:54

Table of Contents

1 HTTP 3

1.1 POST 3

1.2 GET 3

1.3 Biblioteca requests 4

1.4 Exemplo prático 4

1.5 Pegando dados do Firebase 6

1.6 Desafio 6

1 HTTP

Hypertext Transfer Protocol: protocolo de comunicação (na camada de aplicação do Modelo OSI) base para a comunicação de dados da World Wide Web (www).

O HTTP funciona como um protocolo de requisição-resposta no modelo computacional cliente-servidor. Um navegador web, por exemplo, pode ser o cliente e uma aplicação em um computador que hospeda um sítio da web pode ser o servidor. O cliente submete uma mensagem de requisição HTTP para o servidor O servidor retorna uma mensagem resposta para o cliente. A resposta contém informações de estado completas sobre a requisição e pode também conter o conteúdo solicitado no corpo de sua mensagem.

Por exemplo, você pode usar o cliente (browser) para pesquisar uma imagem de 'cachorro' no Google. Então uma request HTTP será enviada para o servidor, que é um local onde a foto está armazenada, o servidor enviará uma resposta com o código de status e o conteúdo solicitado. Esse processo é conhecido como ciclo request-response. Para entender melhor: What is HTTP?

Ao fazer uma requisição HTTP, o cliente deve especificar uma URL Uniform Resource Locator composta pela identificação do protocolo, pelo endereço do computador servidor e pelo documento requisitado.

HTTP define oito métodos: **GET**, **POST** , *HEAD*, *PUT*, *DELETE*, *TRACE*, *OPTIONS* e *CONNECT*. Que indicam a ação requisitada pelo cliente ao servidor. As mais usadas são **GET** e **POST**.

1.1 POST

É usado quando o cliente deseja enviar dados que serão processados no servidor. O uso mais comum é em formulários HTML.

Os parâmetros enviados por uma POST request não ficam visíveis na URL, mas sim no corpo da mensagem. Por isso a preferência pelo POST em formulários, para que dados de usuários não fiquem expostos na URL, onde qualquer um pode ver.

Ex:

```
POST http:// 191.232.179.240:3001/insert/variable                                    -> URL do Servidor
Content-Type: application/json                                                       -> Cabeçalho (Header)
{"id":167180,"value":" 25.534019, 49.32371","timestamp":"2019 09 03T13:16:40.094Z"} -> Pacote de dados
```

Resposta do servidor:

Código com o status da execução da request. Códigos mais comuns:

- 200: Requisição executada com sucesso;
- 400: Erro no cliente;
- 500: Erro no servidor.

1.2 GET

O método GET é utilizado quando o cliente deseja requisitar dados ao servidor, é o método que vamos usar para recuperar os dados do Firebase.

Os parâmetros enviados por uma GET request são anexados junto com a URL, o nome da variável vem sempre depois de um caracter **?** e são separados pelo caracter **&** .

Ex:

```
GET https://www.example.com/index.html?name1=value1&name2=value2
```

A resposta do cliente seria o próprio arquivo de dados que está armazenado naquele endereço do servidor.

Vamos entender como utilizar esses métodos utilizando Python.

1.3 Biblioteca requests

Utilizando a biblioteca requests, nosso programa estará atuando como o cliente HTTP, ou seja, vai fazer o mesmo que o browser faz: enviar requests para servidores, receber os dados da página escolhida e retornar isso para o usuário. A única diferença é que o browser consegue renderizar o arquivo HTML recebido, mostrando visualmente a página para o usuário. Nosso cliente python não fará isso, somente receberá os dados brutos, ou seja, fazendo uma request de alguma página da web, um código HTML será retornado.

O primeiro passo é importar a biblioteca:

```
1  import requests
2  import json     # será útil mais para frente
```

Vamos fazer um request de GET em alguma página e ver o que será retornado:

```
1  r = requests.get('https://www.youtube.com/')
```

O firewall da intranet da Bosch não permite a execução entre essas interfaces, para passar por isso precisamos definir nosso proxy, adicionando ao parâmetro "proxies":

```
1  proxies = {'https': https://<SEU_ID>:<SUA_SENHA>@rb-proxy-de.bosch.com:8080}
2  r = requests.get('https://www.youtube.com/', proxies=proxies)
```

Ao printar **r** a seguinte mensagem deve aparecer:

"<Response [200]>"

Isso significa que a request foi executada com sucesso. Para ver o conteúdo dos nossos dados utilizamos a propriedade "content":

```
1  print(r.content)
```

Então podemos ver o código HTML.

Mas não é muito interessante fazer requests para pegar códigos HTML, geralmente o que estamos procurando são dados, então precisamos acessar os servidores que nos disponibilizem esses dados.

Existem diversas APIs onde podemos acessar dados através de requests, você pode ver uma lista de Free APIs nas quais não é necessária uma chave de autenticação (geralmente obtida através de cadastro, podendo ser pago), ou seja, são dados que estão soltos nesses locais, só esperando para nós irmos pegá-los.

1.4 Exemplo prático

Vamos acessar a API Random Dogs. Enviando um GET request para essa API vamos receber uma imagem aleatória de um banco de imagens de cachorros.

Crie um novo programa, importando as bibliotecas **requests** e **json**, e definindo o seu **proxy** .

```
import requests
import json

proxies = {'https': "https://<SEU_ID>:<SUA_SENHA>@rb-proxy-de.bosch.com:8080"}
```

Agora vamos fazer um GET para pegar o arquivo "woof.json", este arquivo nos dará uma URL contendo a imagem aleatória:

```
url = 'https://random.dog/woof.json'
img_data = requests.get(url,proxies=proxies).content
```

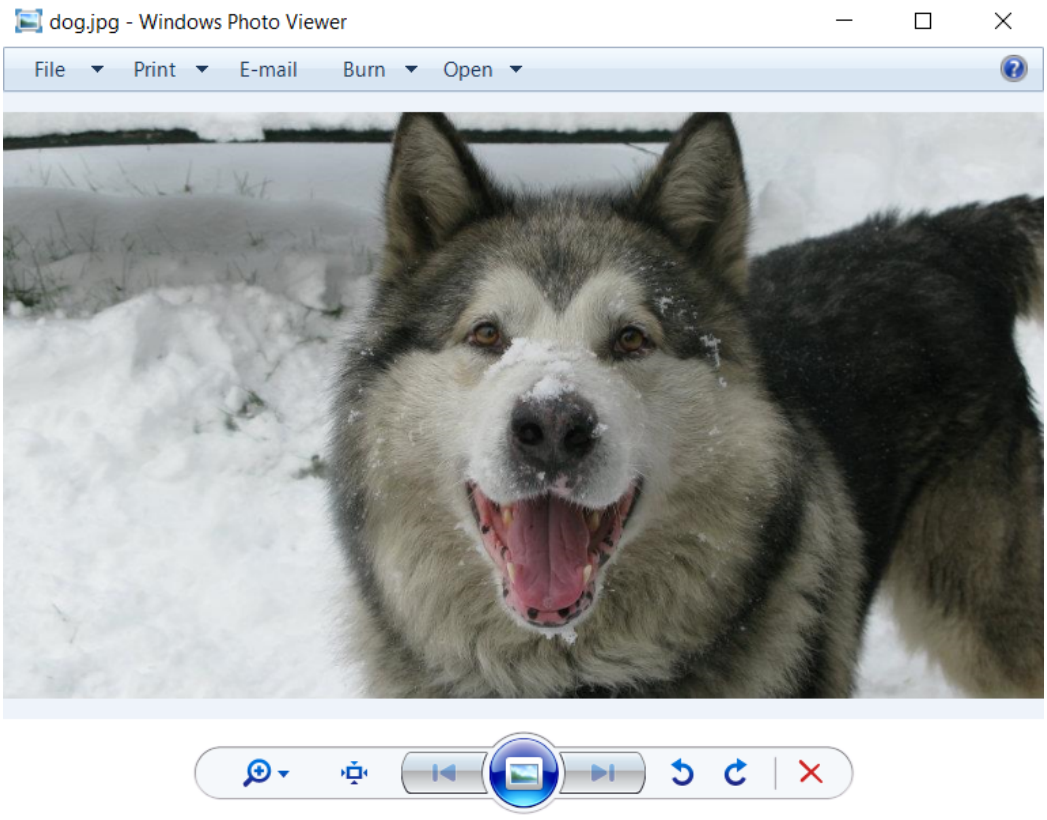
Nossa variável **img_data** é um json, então vamos usar a biblioteca para pegar somente o URL, e então fazer um novo GET request nessa nova URL:

```
new_url = json.loads(img_data)['url']
img = requests.get(new_url,proxies=proxies).content
```

Pronto! O conteúdo da variável **img** é a nossa imagem. Para podermos visualizá-la, vamos salvar como um **jpg** :

```
with open('dog.jpg','wb') as dog_file:
    dog_file.write(img)
```

Ao rodar o programa completo, uma imagem de algum cachorro deve ser salva no mesmo diretório do seu programa:



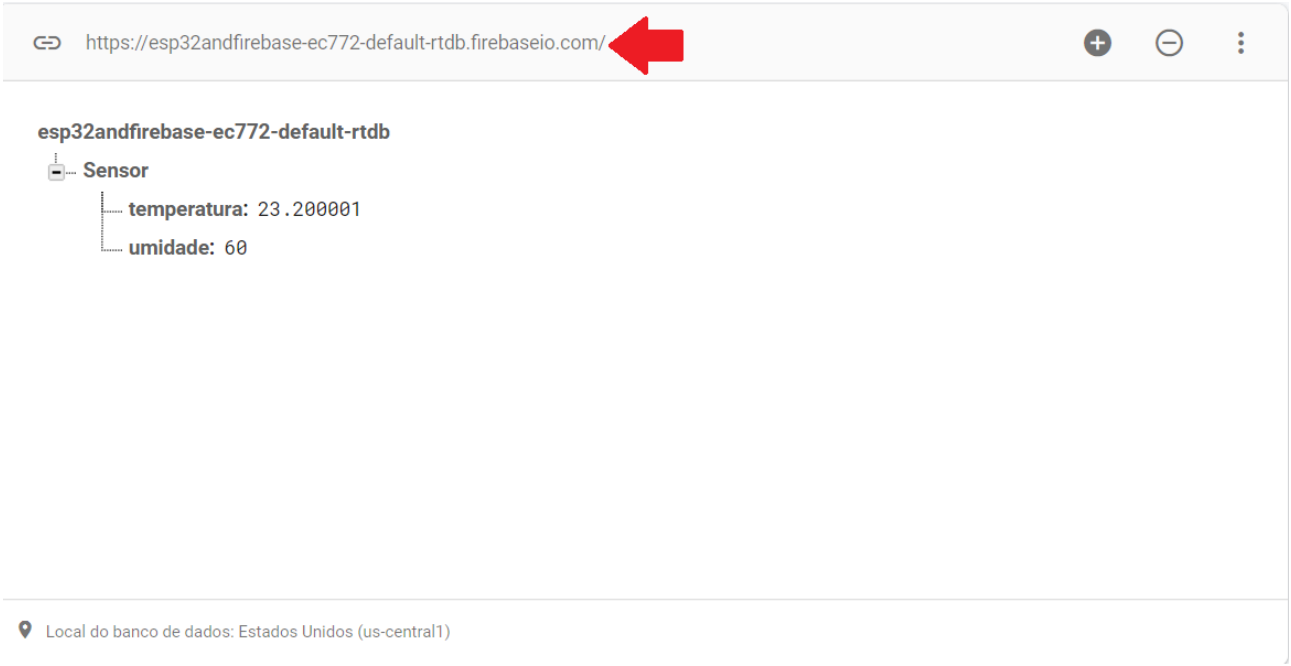
Voltando à nossa aplicação, é a partir desses

requests que iremos pegar os dados que estão no Firebase.

1.5 Pegando dados do Firebase

Assim como as imagens dos cachorros estavam disponíveis e pudemos pegar através de uma URL, os dados do nosso Database também estão disponíveis. Eles estão salvos como json, mas podemos acessar cada variável separadamente.

Vamos utilizar o endereço do nosso Database:




```
1  url_temperatura = 'https://esp32andfirebase-ec772-default-rtdb.firebaseio.com/Sensor/
   temperatura.json'
2  url_umidade = 'https://esp32andfirebase-ec772-default-rtdb.firebaseio.com/Sensor/umidade.json'
```

Com as duas URLs definidas, é só criar as requests:

```
1  temperatura = float(requests.get(url_temperatura, proxies=proxies).content)
2  umidade = float(requests.get(url_umidade, proxies=proxies).content)
```

Sempre que as requests forem executadas, os valores em tempo real do Database serão retornados, ou seja, os últimos valores que a ESP enviou para o servidor do Firebase.

1.6 Desafio

- Faça um programa utilizando a API https://agify.io/, faça um GET request enviando como parâmetro o nome de uma pessoa, o dado retornado deve ser a previsão de idade para a pessoa de acordo com seu nome.