Laboratório 4: Serviços Web em Python

02/03/2021

O objetivo deste laboratório é mostrar como desenvolver um Serviço Web REST em Python e acessar o mesmo por meio do aplicativo `curl`. É também objeto desse laboratório, apresentar como documentar a API do serviço de acordo com a especificação API Blueprint[a].

[a] https://apiblueprint.org/

Sumário

1 Preparando ambiente e instalando pacotes Python

```
mkdir projeto-rest
cd projeto-rest
python3 -m venv venv
source venv/bin/activate
pip install flask
pip install flask-httpauth
```

2 Serviço para gestão de livros

O Listagem 1 apresenta um serviço REST em Python que permite interagir com uma lista de livros armazenada em memória. O recurso `/livrosautenticado` foi criado para demonstrar como exigir a autenticação do usuário ao acessar um recurso. O serviço REST permite:

- Listar todos os livros na lista;
- Consultar dados de um único livro;
- Alterar dados de um livro;
- Excluir um livro da lista.

Na Listagem 1 acima de cada função, na forma de comentário, são apresentadas sintaxes do aplicativo **curl** para interagir com cada recurso. O código fonte apresentado abaixo pode ser obtido em http://docente.ifsc.edu.br/mello/std/labs/python-REST/v1.py.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask, jsonify
from flask import abort
from flask import make_response
from flask import request
from flask import url_for
from flask_httpauth import HTTPBasicAuth

auth = HTTPBasicAuth()

app = Flask(__name__)

livros = [
          {
          'id' : 1,
          'titulo' : 'Linguagem de Programacao C',
          'autor' : 'Dennis Ritchie'
          },
          {
          'id' : 2,
          'titulo' : 'Java como programar',
          'autor' : 'Deitel & Deitel'
          }
]

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/livros
#
@app.route('/livros', methods=['GET'])
def obtem_livros():
    return jsonify({'livros': livros})

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/livros/1
```

```
#
@app.route('/livros/<int:idLivro>', methods=['GET'])
def detalhe_livro(idLivro):
    resultado = [resultado for resultado in livros if resultado['id'] == idLivro]
    if len(resultado) == 0:
        abort(404)
    return jsonify({'livro': resultado[0]})

# Como invocar na linha de comando
#
# curl -i -X DELETE http://localhost:5000/livros/2
#
@app.route('/livros/<int:idLivro>', methods=['DELETE'])
def excluir_livro(idLivro):
    resultado = [resultado for resultado in livros if resultado['id'] == idLivro]
    if len(resultado) == 0:
        abort(404)
    livros.remove(resultado[0])
    return jsonify({'resultado': True})

# Como invocar na linha de comando
#
# curl -i -H "Content-Type: application/json" -X POST -d '{"titulo":"O livro","autor":"Joao"}' http://
    localhost:5000/livros
#
@app.route('/livros', methods=['POST'])
def criar_livro():
    if not request.json or not 'titulo' in request.json:
        abort(400)
    livro = {
        'id': livros[-1]['id'] + 1,
        'titulo': request.json['titulo'],
        'autor': request.json.get('autor', "")
    }
    livros.append(livro)
    return jsonify({'livro': livro}), 201

# Como invocar na linha de comando
#
# curl -i -H "Content-Type: application/json" -X PUT -d '{"titulo":"Novo Titulo"}' http://localhost:5000/
    livros/2
#
@app.route('/livros/<int:idLivro>', methods=['PUT'])
def atualizar_livro(idLivro):
    resultado = [resultado for resultado in livros if resultado['id'] == idLivro]
    if len(resultado) == 0:
        abort(404)
    if not request.json:
        abort(400)
    if 'titulo' in request.json and type(request.json['titulo']) != str:
        abort(400)
    if 'autor' in request.json and type(request.json['autor']) is not str:
        abort(400)
    resultado[0]['titulo'] = request.json.get('titulo', resultado[0]['titulo'])
    resultado[0]['autor'] = request.json.get('autor', resultado[0]['autor'])
    return jsonify({'livro': resultado[0]})

#### Autenticacao simples ####
# Como invocar na linha de comando
#
# curl -u aluno:senha123 -i http://localhost:5000/livrosautenticado
#
@app.route('/livrosautenticado', methods=['GET'])
@auth.login_required
def obtem_livros_autenticado():
    return jsonify({'livros': livros})

# Autenticacao simples
@auth.get_password
def get_password(username):
    if username == 'aluno':
        return 'senha123'
    return None
```

```
@auth.error_handler
def unauthorized():
    return make_response(jsonify({'erro': 'Acesso Negado'}), 403)
################################

# Para apresentar erro 404 HTTP se tentar acessar um recurso que nao existe
@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'erro': 'Recurso Nao encontrado'}), 404)


if __name__ == "__main__":
    print("Servidor no ar!")
    app.run(host='0.0.0.0', debug=True)
```

3 Serviço para listar os arquivos presentes no servidor

O Listagem 2 ilustra um exemplo de como listar todos os arquivos do diretório corrente, onde está sendo executado o servidor, e apresenta uma solução para apresentar a URI para mais detalhes sobre cada arquivo, permitindo assim navegar pelos estados por meio dos *links*. O código fonte pode ser obtido em http://docente.ifsc.edu.br/mello/std/labs/python-REST/v2.py.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask, jsonify
from flask import abort
from flask import make_response
from flask import request
from flask import url_for
from flask import send_file

import os

app = Flask(__name__)

### Como listar arquivos em um diretorio e em seus subdiretorios ###

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/listaarquivos
#
@app.route('/listaarquivos', methods=['GET'])
def obtem_arquivos():
    lista = []
    #https://docs.python.org/2/library/os.html
    for root, dirs, files in os.walk('.'):
        for nome in files:
            linha = {}
            linha['nome'] = nome
            linha['tipo'] = 'arquivo'
            linha['caminho'] = root
            lista.append(linha)
    return jsonify({'arquivos': lista}), 201

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/listaarquivos/{extensao-desejada}
# curl -i http://localhost:5000/listaarquivos/pdf
#
@app.route('/listaarquivos/<extensao>', methods=['GET'])
def obtem_arquivos2(extensao):
    lista = []
    #https://docs.python.org/2/library/os.html
    for root, dirs, files in os.walk('.'):
        for nome in files:
            if nome.endswith(extensao):
```

```
                path = os.path.join(root, nome)
                size = os.stat(path).st_size # in bytes
                linha = {}
                linha['tamanho'] = size
                linha['caminho'] = root
                linha['nome'] = nome
                lista.append(linha)
    return jsonify({'arquivos': lista}), 201



# Como invocar na linha de comando
#
# curl -i http://localhost:5000/detalhesarquivo/nome.extensao
#
@app.route('/detalhesarquivo/<busca>', methods=['GET'])
def detalhe_arquivo(busca):
    lista = []
    #https://docs.python.org/2/library/os.html
    for root, dirs, files in os.walk('.'):
        for nome in files:
            path = os.path.join(root, nome)
            size = os.stat(path).st_size # in bytes
            linha = {}
            linha['tamanho'] = size
            linha['caminho'] = root
            linha['nome'] = nome
            lista.append(linha)
    resultado = [resultado for resultado in lista if resultado['nome'] == busca]
    if len(resultado) == 0:
        abort(404)
    return jsonify({'arquivo': resultado[0]})

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/detalhesarquivo
#
@app.route('/detalhesarquivo', methods=['GET'])
def lista_arquivos():
    lista = []
    for root, dirs, files in os.walk('.'):
        for nome in files:
            path = os.path.join(root, nome)
            size = os.stat(path).st_size # in bytes
            linha = {}
            linha['tamanho'] = size
            linha['caminho'] = root
            linha['nome'] = nome
            lista.append(linha)
    return jsonify({'arquivos': [tornar_caminho_navegavel(arquivo) for arquivo in lista]})

# tornando os links navegaveis
def tornar_caminho_navegavel(arquivo):
    novo_arquivo = {}
    for field in arquivo:
        if field == 'nome':
            novo_arquivo['uri'] = url_for('detalhe_arquivo', busca=arquivo['nome'], _external=True)
        novo_arquivo[field] = arquivo[field]
    return novo_arquivo

# Como invocar (Use seu navegador web, p.e. Firefox)
#
# http://localhost:5000/obterarquivo/{caminho/nome.extensao}
# http://localhost:5000/obterarquivo/foto.png
#
@app.route('/obterarquivo/<path:filename>')
def download_file(filename):
  if '..' in filename or filename.startswith('/'):
    abort(404)
  return send_file(filename,None,False) # True envia como anexo e False como embutido.

# Para apresentar erro 404 HTTP se tentar acessar um recurso que nao existe
@app.errorhandler(404)
```

```
def not_found(error):
    return make_response(jsonify({'error': 'Not found'}), 404)

if __name__ == "__main__":
    print("Servidor no ar!")
    app.run(host='0.0.0.0', debug=True)
```

4 Serviço para apresentação remota de PDFs

Imagine que exista uma Raspberry PI conectada a um projetor multimídia e o objetivo de serviço é permitir a pessoa apresentar *slides*, de um documento PDF, somente consumindo o serviço REST que está em execução na Raspberry PI. Na listagem abaixo são apresentados os recursos que seriam consumidos para abrir um PDF e trocar de páginas.

```
1 # abrindo o PDF
2 curl http://localhost:5000/xpdf/aula-std.pdf
3
4 # indo para a página 2 do arquivo PDF
5 curl http://localhost:5000/xpdf/aula-std.pdf/2
```

O Listagem 3 mostra como invocar um aplicativo externo fazendo:

1. uma chamada não bloqueante – usada para abrir o aplicativo XPDF e deixá-lo em *background*.
2. uma chamada bloqueante – usada para invocar o aplicativo XPDF para trocar de página e o mesmo será encerrado logo após realizar essa ação.

Obtenha o código fonte em http://docente.ifsc.edu.br/mello/std/labs/python-REST/v3.py.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from flask import Flask, jsonify
from flask import abort
from flask import make_response
from flask import request
import subprocess
import os

app = Flask(__name__)
# Como invocar na linha de comando
# Invocando o leitor de PDFs xpdf tem um bug no ubuntu
# Editar sudo vi /usr/bin/xpdf +27 e acrescenter -exec no case
# -z|-g|-geometry|-remote|-rgb|-papercolor|-eucjp|-t1lib|-ps|-paperw|-paperh|-upw|-exec)
#
# curl -i http://localhost:5000/xpdf/{arquivo.pdf}
#
@app.route('/xpdf/<nome>', methods=['GET'])
def abre_pdf(nome):
    # Chamada de processo nao bloqueante - fica em background
    parametro = '-remote projetor ' + nome
    p1 = os.spawnlp(os.P_NOWAIT,"xpdf","xpdf",parametro)
    return jsonify({'resultado': True}), 201

# Como invocar na linha de comando
#
# curl -i http://localhost:5000/xpdf/{arquivo.pdf}/2
#
@app.route('/xpdf/<nome>/<int:pagina>', methods=['GET'])
def avanca_paginas(nome,pagina):
    pagina = 'gotoPage('+str(pagina)+')'
    # chamada de processo bloqueante, porem o xpdf termina logo apos a execucao
    subprocess.call(['xpdf','-remote','projetor','-exec',pagina])
    return jsonify({'resultado': True}), 201

# Para apresentar erro 404 HTTP se tentar acessar um recurso que nao existe
@app.errorhandler(404)
```

```
def not_found(error):
    return make_response(jsonify({'erro': 'Recurso Nao encontrado'}), 404)

if __name__ == "__main__":
    print("Servidor no ar!")
    app.run(host='0.0.0.0', debug=True)
```

5 Exercício

Documente as APIs de todos os serviços apresentados nesse laboratório de acordo com a especificação API Blueprint[1].

- **Sugestão:** Faça uso do editor *Visual Studio Code* (tem instalado no laboratório) com as extensões *API Elements extension* e *API Blueprint Viewer* (vocês possuem permissão para instalar);
- Em https://apiblueprint.org/documentation/ tem uma documentação sobre a API Blueprint.;
- Em laboratorios.apib tem um exemplo de documentação para o serviço de gestão de laboratórios, desenvolvido no laboratório REST com Java;
- Em https://laboratorios.docs.apiary.io tem a renderização da documentação da API do serviço de gestão de laboratórios, usando a plataforma Apiary[2].

DESAFIO: Faça uso do Dredd[3], framework para teste de APIs REST, para verificar se a documentação está de acordo com um serviço implantado. Dredd lê sua API documentada (p.ex. arquivo .apib) e executa passo a passo em um servidor onde tem essa API implementada. Verifica-se aqui se a sua implementação está gerando as respostas descritas na documentação de sua API.

Referências

1. http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
2. http://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask
3. http://blog.miguelgrinberg.com/post/designing-a-restful-api-using-flask-restful

[1] https://apiblueprint.org/
[2] https://apiary.io
[3] https://dredd.readthedocs.io/