Webex Teams Hackathon 2018

Lab1 - Creating REST API Back End in Python

Objectives

In this lab, you will complete the following objectives:

- Create a simple REST API back end using Flask
- Create a simple GET API endpoint
- Create endpoint serving JSON response
- Create POST API endpoint
- Use browser and Postman to test the endpoints

Background / Scenario

Python can be used to act as a REST API endpoint serving GET, POST and other API requests. In this lab, you will install the Flask module. Flask is a microframework for Python which can be used for serving API endpoints.

When completed, the **lab1-back-end-python.py** program will run as a REST API server and serve:

- simple GET API endpoint
- endpoint which sends back JSON response
- simple POST API endpoint which can modify the data stored in the back end system.

Required Resources

- Postman application
- Python 3 with IDLE
- Python code files

Step 1: Create your Flask project

In this step, you will create your project, install Flask and import Flask to your project.

a. Create your project directory called `lab1-back end-python` and create a `lab1-back-end-python.py` file in it.

b. Open a command line in administrative mode and install Flask using the following command:
`pip install Flask`

c. Edit your `lab1-back-end-python.py` file and import Flask. Add the following text to the beginning of the file:
`from flask import Flask`

Step 2: Create and test your first API endpoint

a. Create a Flask HTTP back end: `app = Flask(__name__)`

b. Add your first GET endpoint:

```
@app.route("/api/helloworld")
      def hello():
```

```
    return "Hello World!"
```

This endpoint can be accessed by the following URL:
http://<your_server_name>/api/helloworld

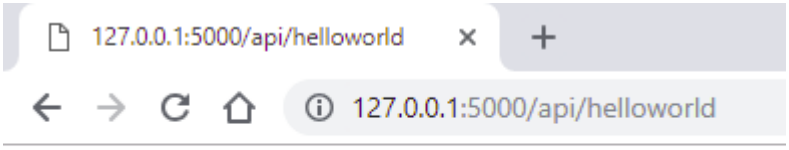
Note: It is not mandatory but recommended to use the '/api' prefix for your REST API endpoints. In your real project you may use whatever prefix you want.

c. Add `app.run()` command to the end of the file to run your Flask HTTP back end instance.

d. Launch your back end app. In the command line type `python lab1-back-end-python.py`

```
* Serving Flask app "backend" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

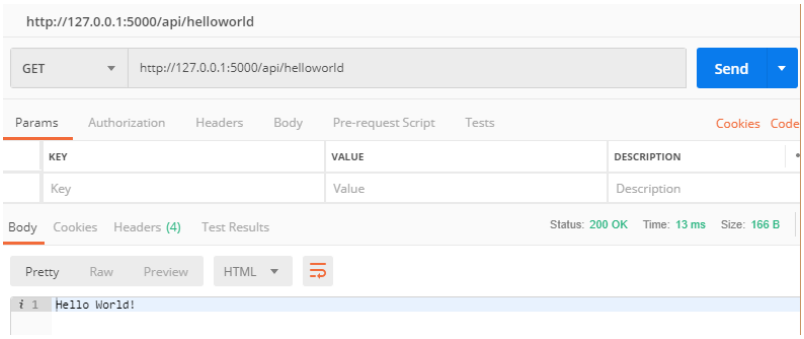
e. Open your Chrome browser and type the following url: http://127.0.0.1:5000/api/helloworld




```
* Serving Flask app "backend" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [02/Feb/2019 11:52:55] "GET /api/helloworld HTTP/1.1" 200 -
```

Congratulation, you have created your first REST API endpoint!

f. Test your endpoint in Postman



g. Stop your back end app. Press `Crtl-C` in the command windows.

Step 3: Create endpoint serving JSON response

a. Import `jsonify` module from Flask. Change your Flask import line for the following:

```
from flask import Flask, jsonify
```

b. Add these variables before your endpoint created at Step 2:

```
name = "Charles Webex"
age = "15"
```

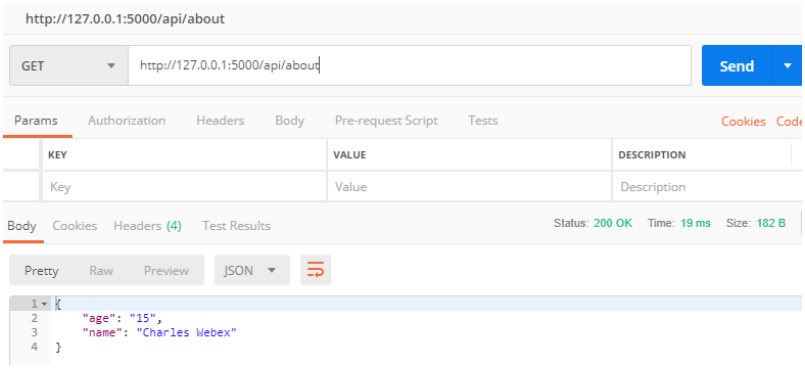
c. Create a new endpoint:

```
@app.route("/api/about")
def about():
      return jsonify(name = name, age = age)
```

d. Launch your back end app. In the command line type `python lab1-back-end-python.py`

e. Open your Chrome browser and type the following url: http://127.0.0.1:5000/api/about

f. Test your endpoint in Postman:



g. Stop your back end app. Press `Crtl-C` in the command windows.

Step 2: Create POST API endpoint

a. Import `request` module from Flask. Change your Flask import line for the following:

```
from flask import Flask, jsonify, request
```

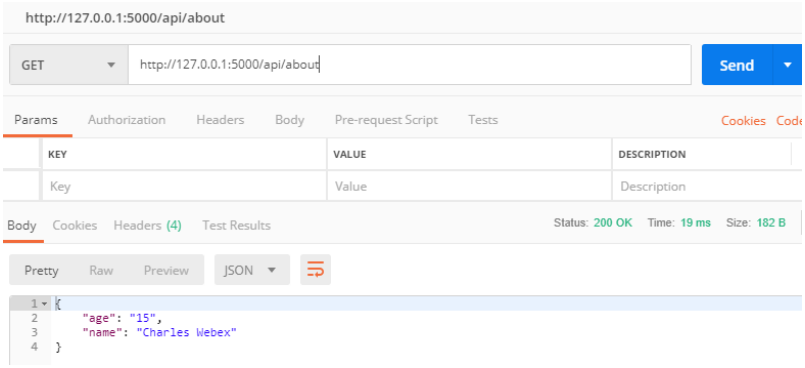
b. Modify your endpoint created at Step 3:

```
@app.route("/api/about", methods = ['POST', 'GET'])
def about():
      global name, age
      if request.method == 'GET':
            return jsonify(name = name, age = age)
      elif request.method == 'POST':
            r = request.json
            name = r["name"]
            age = r["age"]
            return jsonify(name = name, age = age)
```

Note: we use `global` command because we would like to use the global `name` and age `variable`. Without `global` command a new local (in the scope of `about()` function) variable would be created when we assign a new value to `name` or `age` variable.

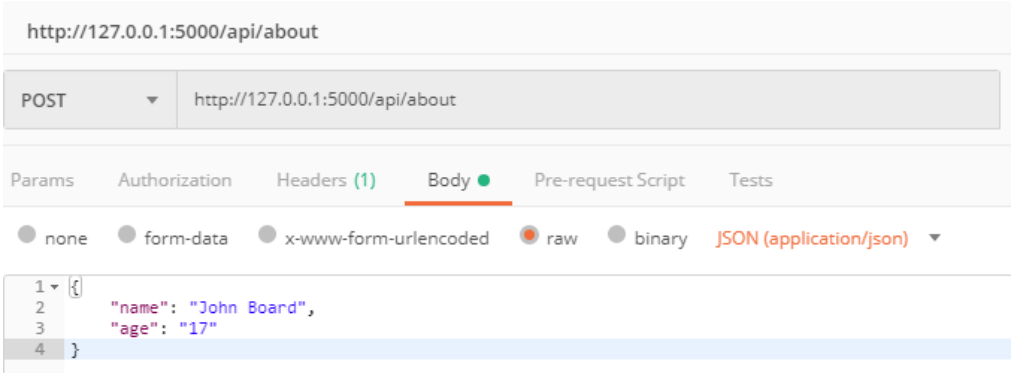
c. Launch your back end app. In the command line type `python lab1-back-end-python.py`

d. Test your GET endpoint in Postman:

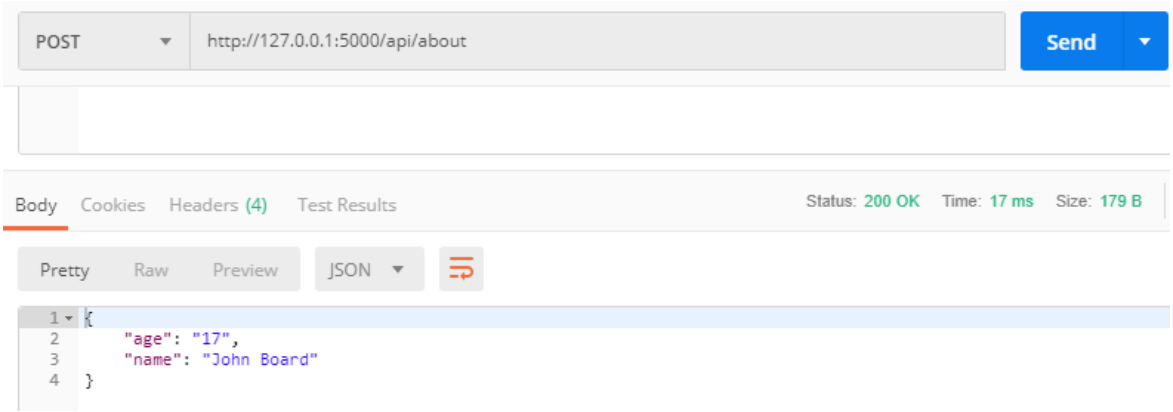


e. Test your POST endpoint in Postman. Use **raw** format and **JSON (application/json)** type in the Body settings then give the following body content:

```
{
      "name": "John Board",
      "age": "17"
}
```



Click Send button to send your request to your back end.



Note: in the response from your back end you can find the name and age values in JSON format.

f. Check the new state using your GET endpoint.