Flask REST API Tutorial

REST API services let you interact with the database by simply doing HTTP requests. In this article you learn how to write a REST server using the Flask.

This is often how the backend of web apps is created. Returning data is in JSON format and requests we are using are PUT, DELETE, POST, and GET

If you want to put your API online, use: PythonAnywhere.

Related course: Python Flask: Create Web Apps with Flask

Flask API example

Introduction

To make our first program, recall that we enter the URL in the browser

```
localhost:5000
```

At the time, a string “Hello World!” was returned, so we thought, can we replace this string with a json sequence? Isn’t that the same as a REST query API?

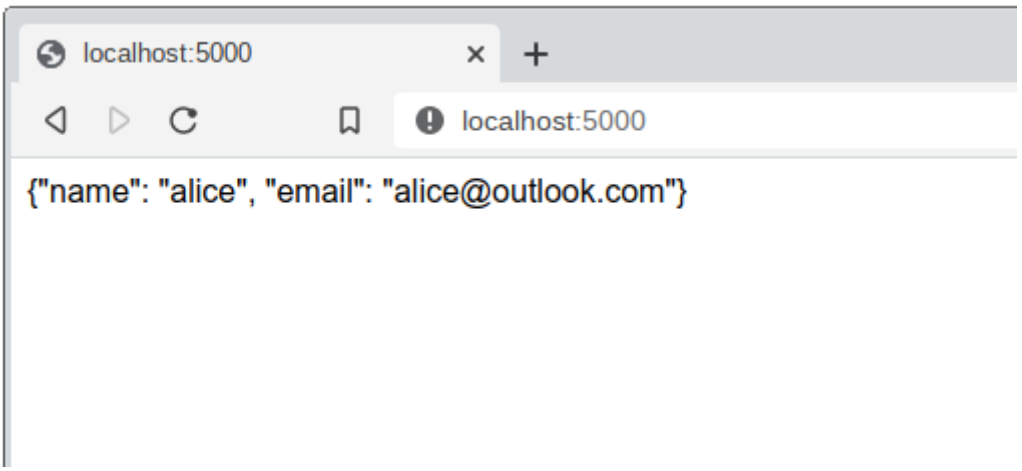
So, we might have the first impulse to do this:

```
#!/usr/bin/env python
# encoding: utf-8
import json
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
```

```
        return json.dumps({'name': 'alice',
                           'email': 'alice@outlook.com'})
app.run()
```

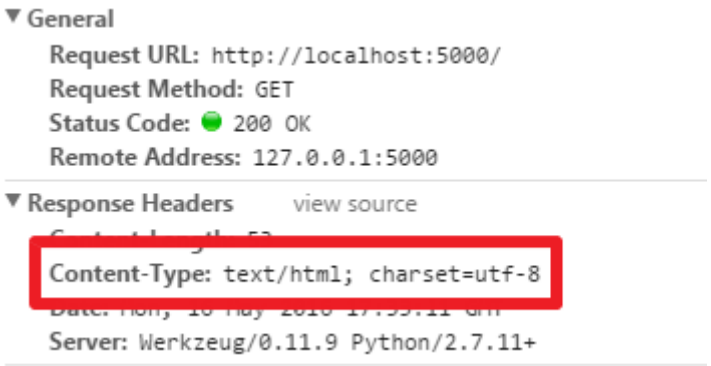
Actually, we just modified the returned string, modified it to a string of JSON, and then we opened it on the browser

```
localhost:5000
```



Wow! It seems to have achieved the function we wanted, returned a JSON string.

But we opened the debug tool for Chrome (which I use as a tool similar to Chrome, Safari, Firefox) (under Windows: Ctrl + Alt + I, Mac under: Cmd + Shift + I), we can see that this returned data type actually is of type html:



> You may wonder what impact this might have, the impact should be small in most cases, but for some mobile-side libraries, the data may be processed according to the response (incorrectly!).

If you want to put your API online, use: PythonAnywhere.

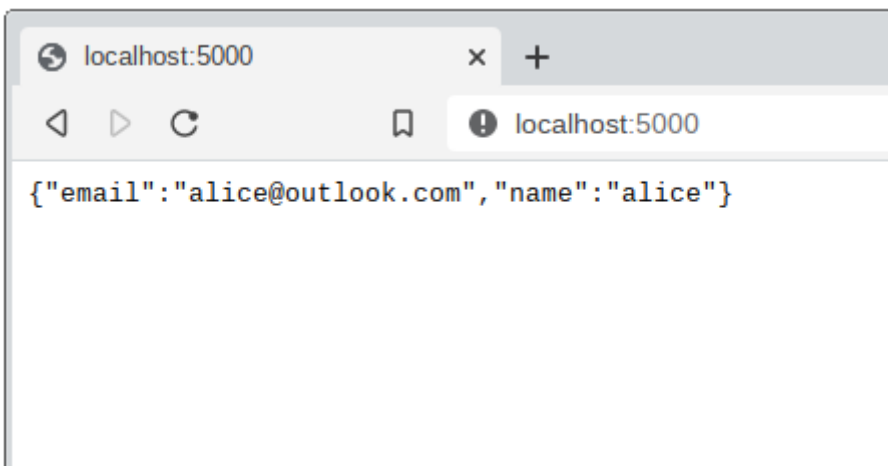
Return json

To deal with this situation, we can't simply set this response head into json format.
A better solution is to use the jsonify function of the Flask, where I use this function to modify the code:

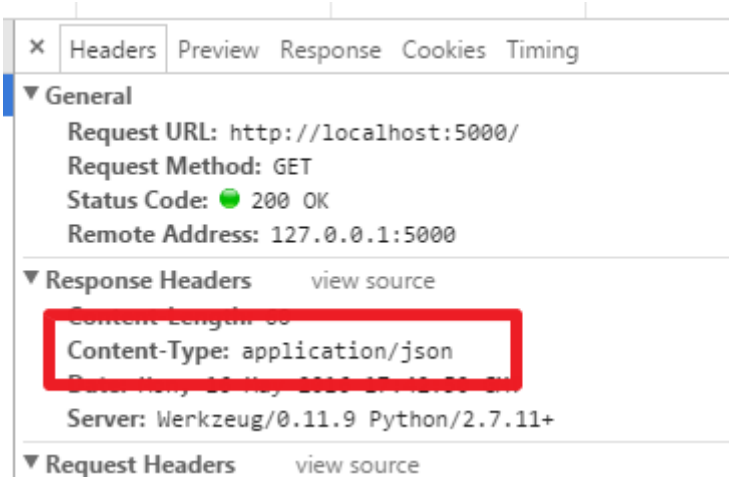
```
#!/usr/bin/env python
# encoding: utf-8
import json
from flask import Flask, jsonify
app = Flask(__name__)
@app.route('/')
def index():
    return jsonify({'name': 'alice',
                    'email': 'alice@outlook.com'})

app.run()
```

The changes are:

```
from flask import ...., jsonify
... ...
    return jsonify({'name': 'alice',
                    'email': 'alice@outlook.com'})
```



Look at Google Dev Tools, you'll see the content-type change to JSON.



Request method

We know that there are six commonly used HTTP request methods, which are

- GET
- POST
- PUT
- DELETE
- PATCH
- HEAD

The code that we just had had to deal with GET by default (the browser defaults to using GET), so how do you program the other requests?

Like this:

```
@app.route('/', methods=['POST'])
@app.route('/', methods=['DELETE'])
@app.route('/', methods=['PUT'])
```

The program below demonstrates this:

```
#!/usr/bin/env python
# encoding: utf-8
import json
```

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/', methods=['GET'])
def query_records():
    name = request.args.get('name')
    print name
    with open('/tmp/data.txt', 'r') as f:
        data = f.read()
        records = json.loads(data)
        for record in records:
            if record['name'] == name:
                return jsonify(record)
        return jsonify({'error': 'data not found'})

@app.route('/', methods=['PUT'])
def create_record():
    record = json.loads(request.data)
    with open('/tmp/data.txt', 'r') as f:
        data = f.read()
    if not data:
        records = [record]
    else:
        records = json.loads(data)
        records.append(record)
    with open('/tmp/data.txt', 'w') as f:
        f.write(json.dumps(records, indent=2))
    return jsonify(record)

@app.route('/', methods=['POST'])
def update_record():
    record = json.loads(request.data)
    new_records = []
    with open('/tmp/data.txt', 'r') as f:
        data = f.read()
        records = json.loads(data)
    for r in records:
        if r['name'] == record['name']:
            r['email'] = record['email']
        new_records.append(r)
    with open('/tmp/data.txt', 'w') as f:
        f.write(json.dumps(new_records, indent=2))
```

```
    return jsonify(record)

@app.route('/', methods=['DELETE'])
def delte_record():
    record = json.loads(request.data)
    new_records = []
    with open('/tmp/data.txt', 'r') as f:
        data = f.read()
        records = json.loads(data)
        for r in records:
            if r['name'] == record['name']:
                continue
            new_records.append(r)
    with open('/tmp/data.txt', 'w') as f:
        f.write(json.dumps(new_records, indent=2))
    return jsonify(record)

app.run(debug=True)
```

The code is long, but the code is easier to understand, and it is a relatively simple file operation.

The code that we need to focus on is the following:

- How to set request methods

```
@app.route('/', methods=['GET'])
@app.route('/', methods=['PUT'])
@app.route('/', methods=['POST'])
@app.route('/', methods=['DELETE'])
```

- How to get data

If you want to put your API online, use: PythonAnywhere.

Back **Next**

Deploy Flask App How to Set Up Flask with MongoDB