Challenge Lab 08.04

Create a REST Server

Description:
Create an application server that will implement a RESTful API.
The API should allow a user to perform CRUD operations on an entity (eg a book)
Test it using CURL

Suggested steps:

- Design the REST API (create a document)
- Create a virtual environment to run a Flask server
- Create a basic Flask Server
- Modify the Flask Server to deal with the required URL mappings (the functions can be just skeletons at this stage
- Implement getAll and test it
- Implement findById and test it
- Implement create and test it
- Implement update and test it
- Implement delete and test it

Design the REST API (create a document)

1. Using the skills you learned in week 05 create a REST API for an entity of your choice.

Create a virtual environment to run a Flask server

2. Create a directory where this server is going to run
3. Create a Virtual environment

```
python -m venv venv
```

4. Run the virtual environment

```
.\venv\Scripts\activate.bat
```

5. Install Flask

```
Pip install flask
```

6. Save the package list in a file called requirements.txt

```
pip freeze > requirements.txt
```

7. Create a .gitignore file and put in venv/

Create a basic Flask Server

8. Create a file called rest_server.py
9. Make a basic server in it and test it

```
from flask import Flask, url_for, request, redirect, abort

app = Flask(__name__, static_url_path='', static_folder='staticpages')

@app.route('/')
def index():
    return "hello"


if __name__ == "__main__":
    app.run(debug=True)
```

Modify the Flask Server to deal with the required URL mappings (the functions can be just skeletons at this stage)

10. Create a mapping and function for each of the possible API calls

```
@app.route('/books')
def getAll():
    return "served by Get All()"
# find By id
@app.route('/books/<int:id>')
def findById(id):
    return "served by find by id with it "+ str(id)

# create
@app.route('/books', methods=['POST'])
def create():
    return "served by Create "

#update
@app.route('/books/<int:id>', methods=['PUT'])
def update(id):
    return "served by update with it " + str(id)

#delete
@app.route('/books/<int:id>', methods=['DELETE'])
def delete(id):
    return "served by delete with it " + str(id)
```

Implement getAll and test it

11. Create an array at the top of the file to store the data
12. Create a variable called nextId
13. Implement the getAll function by returning the array

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

app = Flask(__name__, static_url_path='', static_folder='staticpages')

books=[
    {"id": 1, "Title": "Harry Potter", "Author": "JK", "Price": 1000},
    {"id": 2, "Title": "some cook book", "Author": "Mr Angry Man", "Pri
ce": 2000},
    {"id": 3, "Title": "Python made easy", "Author": "Some Liar", "Pric
e": 1500}
]
nextId=4

#get all
#curl http://127.0.0.1/books
@app.route('/books')
def getAll():
    return jsonify(books)
```

Implement findById and test it

14. Implement the findById function, by searching in the array for an object that has a matching id and return that object

```
#curl http://127.0.0.1/books/1
@app.route('/books/<int:id>')
def findById(id):
    foundBooks = list(filter (lambda t : t["id"]== id, books))
    if len(foundBooks) == 0:
        return jsonify({}) , 204
    return jsonify(foundBooks[0])
```

Implement create and test it

15. Implement the create function, create an object with data in it, and store it in the array, make sure that it does not have an id that already exists (eg use the nextId variable)
16. Return that object
17. Test this

```
# create
# curl -X POST -
d "{\"Title\":\"test\", \"Author\":\"some guy\", \"Price\":123}" http:/
/127.0.0.1:5000/books
@app.route('/books', methods=['POST'])
def create():
    global nextId

    book = {
        "id": nextId,
        "Title": "Test",
        "Author": "Test"
        "Price": 999
    }
    books.append(book)
    nextId += 1
    return jsonify(book)
```

18. Check that the user posted a JSON object (return error if they did not), set the contents of the object to be the values from the JSON object that was uploaded.

```
# create
# curl -X POST -
d "{\"Title\":\"test\", \"Author\":\"some guy\", \"Price\":123}" http:/
/127.0.0.1:5000/books
@app.route('/books', methods=['POST'])
def create():
    global nextId
    if not request.json:
        abort(400)

    book = {
        "id": nextId,
        "Title": request.json["Title"],
        "Author": request.json["Author"],
        "Price": request.json["Price"]
    }
    books.append(book)
```

Implement update and test it

19. Implement the update function.
    a. Find the object in the array that needs to be updated
    b. If the attribute was uploaded then update that attribute
    c. Return the updated object
    d. Test it

```
#update
# curl -X PUT -d "{\"Title\":\"new Title\", \"Price\":999}" -
H "content-type:application/json" http://127.0.0.1:5000/books/1
@app.route('/books/<int:id>', methods=['PUT'])
def update(id):
    foundBooks = list(filter(lambda t: t["id"] == id, books))
    if len(foundBooks) == 0:
        return jsonify({}), 404
    currentBook = foundBooks[0]
    if 'Title' in request.json:
        currentBook['Title'] = request.json['Title']
    if 'Author' in request.json:
        currentBook['Author'] = request.json['Author']
    if 'Price' in request.json:
        currentBook['Price'] = request.json['Price']

    return jsonify(currentBook)
```

Implement delete and test it

20. Implement the Delete function
    a. Find the object you wish to delete and remove it from the array

```
#delete
# curl -X DELETE http://127.0.0.1:5000/books/1
@app.route('/books/<int:id>', methods=['DELETE'])
def delete(id):
    foundBooks = list(filter(lambda t: t["id"] == id, books))
    if len(foundBooks) == 0:
        return jsonify({}), 404
    books.remove(foundBooks[0])

    return jsonify({"done":True})
```