Lab 05.01

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. Creat 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, I have done this for books)

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

```
@app.route('/books', methods=['GET'])
def getall():
        return "get all"

@app.route('/books/<int:id>', methods=['GET'])
def findbyid(id):
        return "find by id"

#create
@app.route('/books', methods=['POST'])
def create():
        # read json from the body
        jsonstring = request.json

        return f"create {jsonstring}"

# update
@app.route('/books/<int:id>', methods=['PUT'])
def update(id):
        jsonstring = request.json
        return f"update {id} {jsonstring}"

#delete
@app.route('/books/<int:id>', methods=['DELETE'])
def delete(id):
        return f"delete {id}"
```

Test them.

11. Using either postman or CURL test each of your endpoints

```
# getall
# curl http://127.0.0.1:5000/books

# find by id
# curl http://127.0.0.1:5000/books/1

#create
#curl -X POST -d "{\"title\":\"test\", \"author\":\"some guy\",
\"price\":123}" http://127.0.0.1:5000/books

# update
# curl -X PUT -d "{\"title\":\"test\", \"author\":\"some guy\",
\"price\":123}" http://127.0.0.1:5000/books/1

#delete
# curl -X DELETE  http://127.0.0.1:5000/books/1
```

Future: create the DAO

12. Create another file for interacting with the database, mine is for books so is called bookdao.py

```
# returns all the books in the database table
def getall():
    return [{}]
# returns one book
def findById(id):
    return {}
# inserts a book into the database
def create(book):
    return book
# updates the book
def update(id, book):
    return book
# deletes the book with the id
def delete(id):
    return True
```

Future: integrate the DAO with the server