COMP3330 Interactive Mobile Application Design and Programming

Tutorial 4

Building a server response system using Python 3 (Flask) & SQLite for processing JSON responses (Part 1)

1. Introduction

In this and the next tutorial, we will use Python & SQLite to build a web server response system for processing responses in JSON format such that the Android apps will be able to get data from the SQL database in an easy manner.

JSON is a syntax for storing and exchanging data which is similar and alternative to XML. JSON data is written as key-value pairs and separated by commas. There are two major types of JSON data: (i) JSON object and (ii) JSON array. For the JSON object, it is held by a pair of curly brackets "{}"; and for the JSON array, it is held by a pair of square brackets "[]". For more details about the JSON syntax, please refer to the JSON tutorial provided by w3schools: https://www.w3schools.com/js/js_json_syntax.asp

2. Setup flask for processing JSON responses

We will build a python web server using the Flask framework. The easiest way to install Flask is by PIP package manager. The latest version of Flask (2.2.x) supports Python 3.7 and newer. For more details about Flask, you may go to their website: https://flask.palletsprojects.com/

You can download Python from its official page: https://www.python.org/

PIP is included by default as of Python version 3.4. If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

When you have Python and PIP installed in your environment, run **py -m pip install flask** or **pip install flask** to install Flask.

Once Flask is installed in your Python environment, we can start using Python to build the server.

2.1. Create a new Python file and save it, name it server.py
2.2. Open the source file with a text/code editor.
2.3. Add the following code to server.py. flask.Flask() is used to create an instance of Flask class. route() is a function decorator that tells Flask what URL should trigger the function. In the code, "/tutorial" will trigger tutorial(). The function returns the message that sends back to the client. If the return is a dictionary, Flask will turn it into a JSON response. app.run() with parameter host="0.0.0.0" runs the server and tells it to listen on all public IPs.

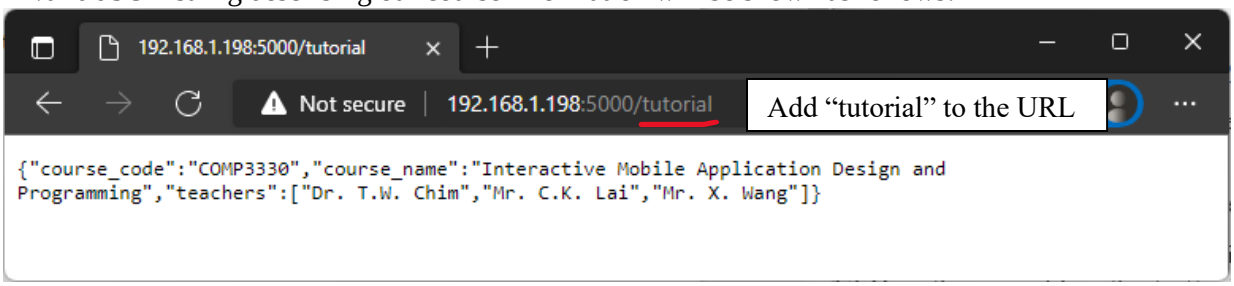
```
import flask

app = flask.Flask(__name__)
#app.config["DEBUG"] = True # Enable debug mode to enable hot-reloader.

@app.route('/tutorial')
def tutorial():
    outdata = {
        "course_code": "COMP3330",
        "course_name": "Interactive Mobile Application Design and Programming",
        "teachers": ["Dr. T.W. Chim", "Mr. C.K. Lai", "Mr. X. Wang"]
    }
    return outdata

# adds host="0.0.0.0" to make the server publicly available
app.run(host="0.0.0.0")
```

2.4. Run the python program (e.g., run **py server.py** in the terminal)

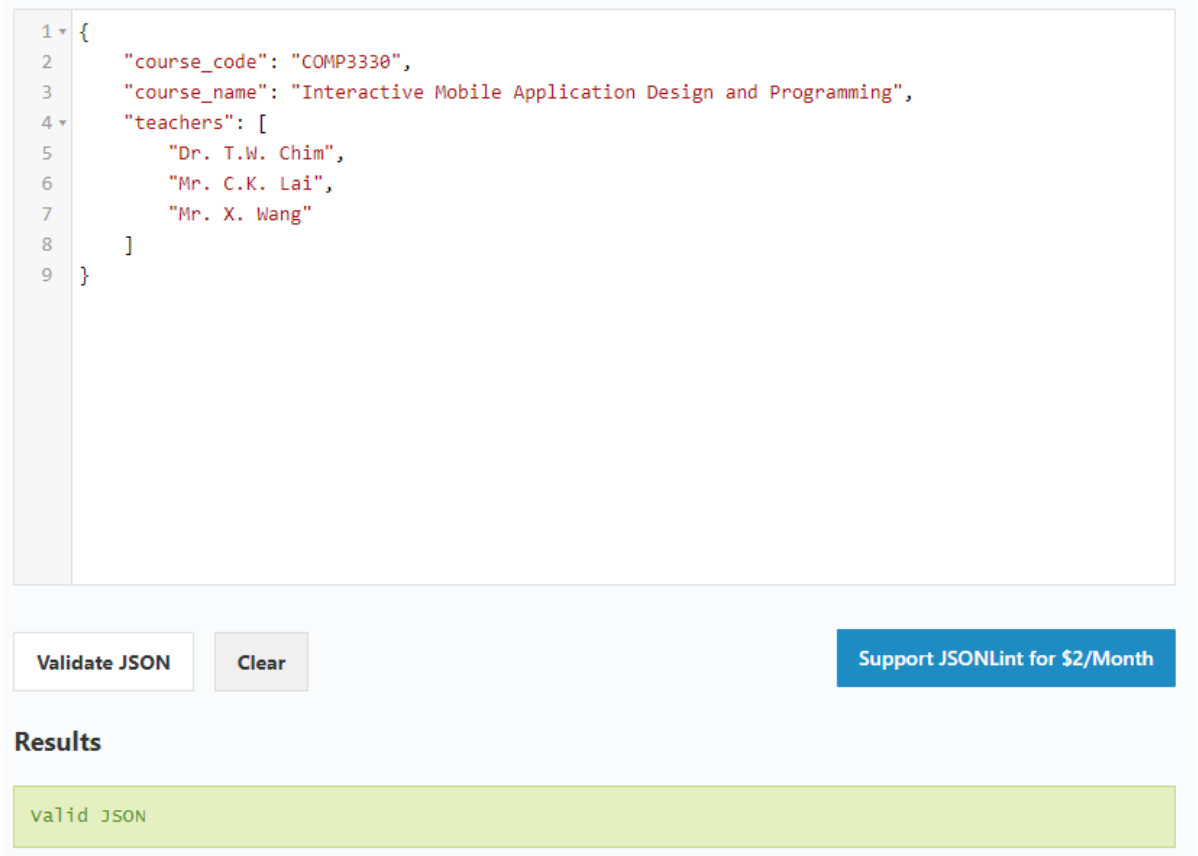
2.5. Open a web browser, and input the address displayed in the terminal and add “tutorial” to the URL.

```
* Running on http://192.168.1.198:5000/
```

A valid JSON string describing our course information will be shown as follows:



2.6. You can use any JSON validator on the Internet to check whether the format of your JSON string is correct, e.g. https://jsonlint.com/



2.7. For more details about the JSON syntax, please refer to the JSON tutorial provided by w3schools: https://www.w3schools.com/js/js_json_syntax.asp

3. Setup SQLite database

We have to set up a database to make the JSON response dynamic. SQLite is a self-contained, file-based SQL database and can be used in your Python program without having to install any additional software. We can use sqlite3 module in Python to connect to an SQLite database.

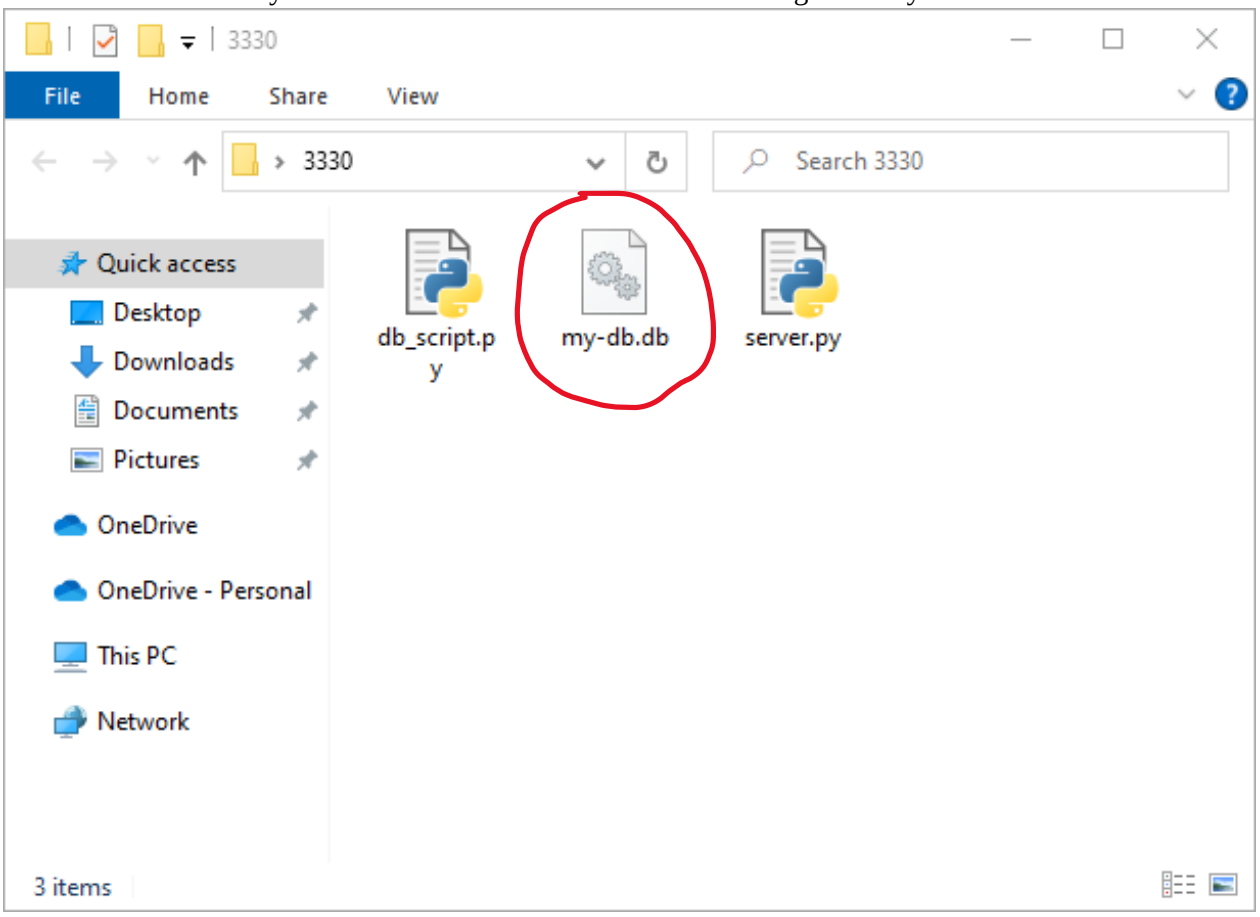
For details about SQLite, please refer to www.sqlite.org. (You don’t need to download anything from this website)

3.1. Run the following Python code to build the database together with a table “STUDENT” and 5 rows of records.

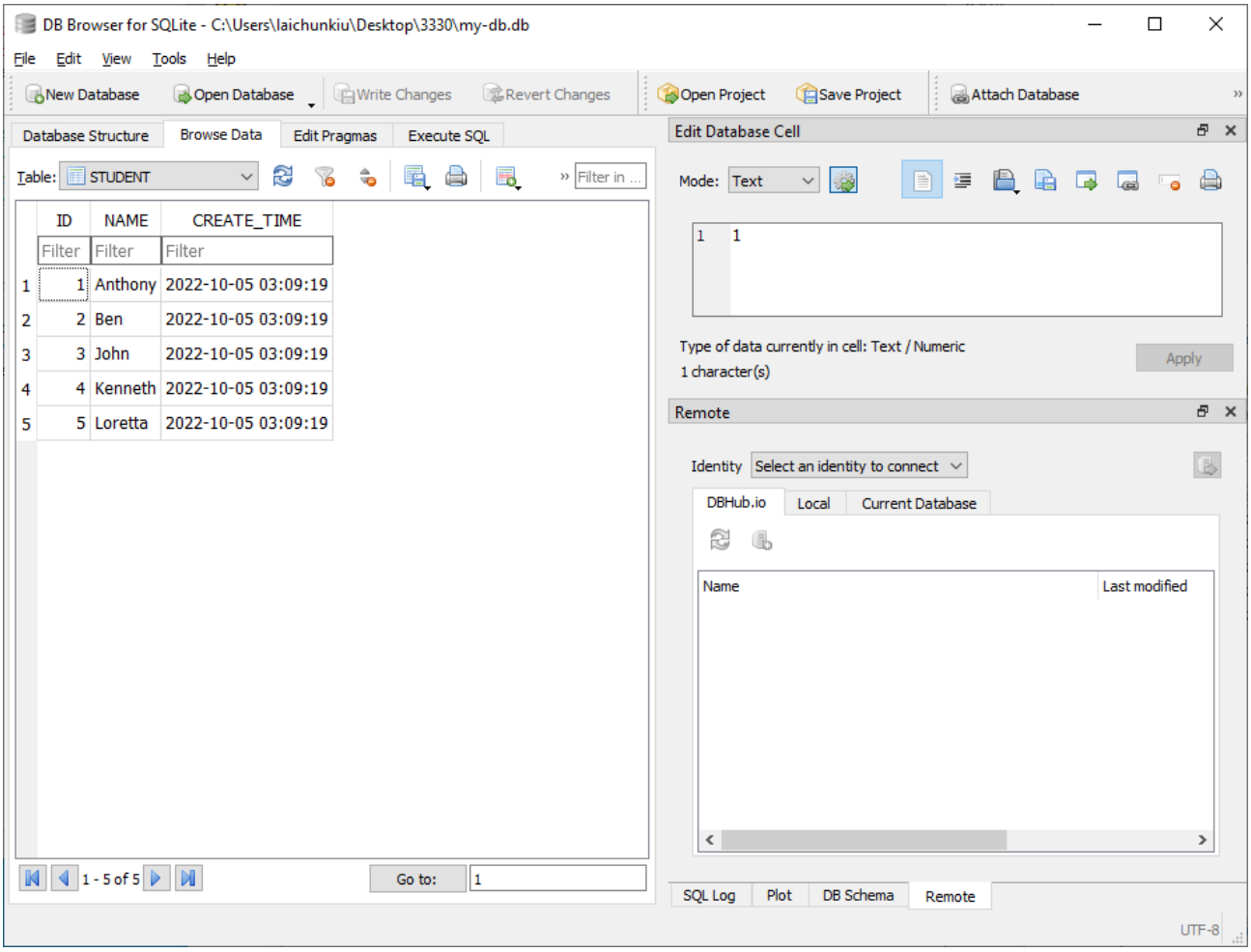
```
import sqlite3

# Connects to database
# The .db file is created automatically if it does not exist
con = sqlite3.connect('my-db.db')
```

```
# Creates table
con.execute("""CREATE TABLE IF NOT EXISTS STUDENT (
    ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME TEXT NOT NULL,
    CREATE_TIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );""")

# insert test data
testData = ['Anthony', 'Ben', 'John', 'Kenneth', 'Loretta']

for name in testData:
    insertQuery = "INSERT INTO STUDENT (NAME) values (?);"
    con.execute(insertQuery, (name,))
con.commit()
con.close()
```

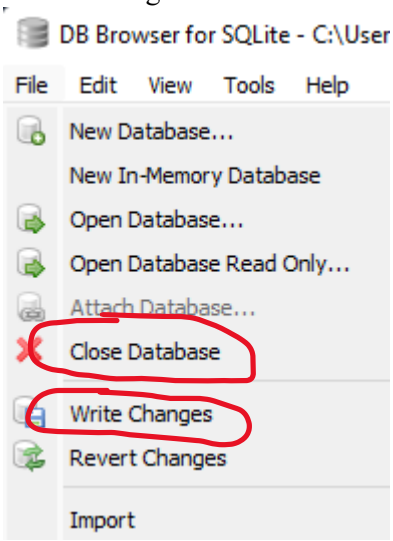
3.2. Then a database file my-db.db will be created in the current working directory.



3.3. (Optional) You can use DB Browser for SQLite to view and edit the database file. You can download it from https://sqlitebrowser.org/.



[Tips] If you have made changes to the database using DB Browser for SQLite, remember to click “File” -> “Write changes” to save file and then “Close Database” before you leave.



4. Select data from SQLite database to Python

4.1. Go back to the server.py, import sqlite3.

```
import sqlite3
```

4.2. Add the following codes to tutorial(). The “SELECT” SQL statement is used to select data from a database. The Python syntax cursor is used to get the returned results from the SQLite side. Finally, the append() stores all data results in a Python list.

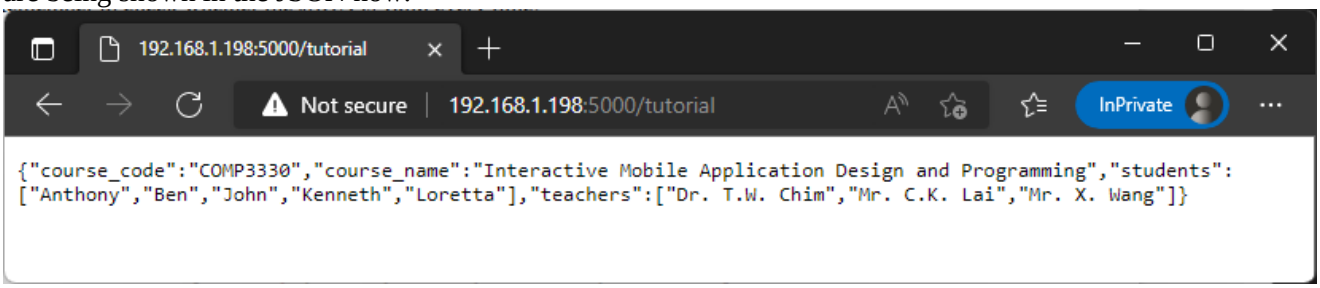
```
    con = sqlite3.connect('my-db.db')
    cursor = con.execute("SELECT NAME from STUDENT;")
```

```
    students = []
    for row in cursor:
        students.append(row[0])

    con.close()
```

4.3. Add a new item to the outdata dictionary, whereas the value refers to the name list we previously loaded from the database. The dictionary will become:

```
    outdata = {
        "course_code": "COMP3330",
        "course_name": "Interactive Mobile Application Design and Programming",
        "teachers": ["Dr. T.W. Chim", "Mr. C.K. Lai", "Mr. X. Wang"],
        "students": students
    }
```

4.4. Restart the Python program and refresh the webpage. You will see that the student names stored in the database are being shown in the JSON now.



5. Insert data from Python to SQLite database

Besides “SELECT” query, the command can include data modification queries such as “INSERT”, “UPDATE”, and “DELETE”, as well as other SQL commands. Here we will try an example of how to “INSERT” data from Python to SQLite. Unlike the code in step 3.1 where the data are stored in Python code, here we receive the data from client’s request via the URL variable. We use the existence of URL variable to separate the SELECT and INSERT operations such that we can insert a new record of data if and only if the URL contains “?name=”.

5.1. Go back to the server.py, import request module.

```
from flask import request
```

5.2. Add parameter methods=['GET'] to the route(). The code will become:

```
@app.route('/tutorial', methods=['GET'])
```

5.3. In tutorial(), add the following code to get the URL variable “name”

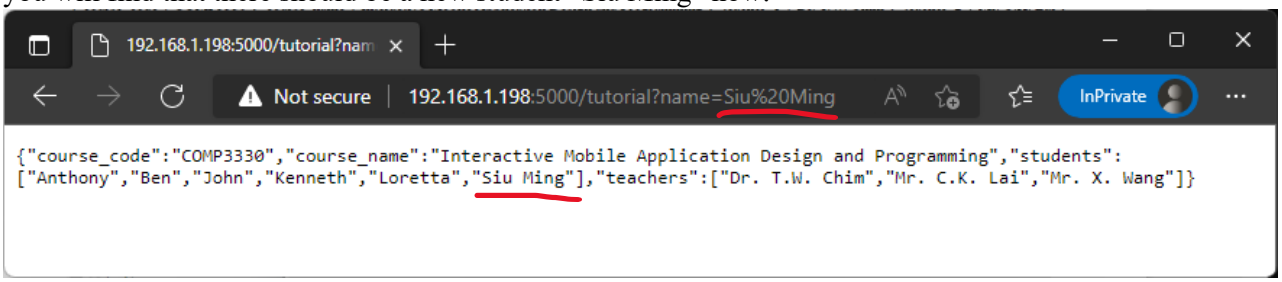
```
    name = request.args.get('name', '')
```

5.4. Next, add logic to separate the SELECT and INSERT operations. The remaining codes of tutorial() will become this. If the length of name > 0, meaning a valid name is received from the client’s request, we insert a new record into the database. Then, we select data from the database as in Step 4.

```
    con = sqlite3.connect('my-db.db')

    if len(name) > 0:
        # insert into db
        insertQuery = "INSERT INTO STUDENT (NAME) values (?);"
        con.execute(insertQuery, (name,))
        con.commit()

    # select from db
```

```
    cursor = con.execute("SELECT NAME from STUDENT;")

    students = []
    for row in cursor:
        students.append(row[0])

    con.close()

    outdata = {
        "course_code": "COMP3330",
        "course_name": "Interactive Mobile Application Design and Programming",
        "teachers": ["Dr. T.W. Chim", "Mr. C.K. Lai", "Mr. X. Wang"],
        "students": students
    }
    return outdata
```

5.5. Open the web browser again, but this time we try to insert a new data record by adding "?name=Siu%20Ming", you will find that there should be a new student "Siu Ming" now.



6. Appendix

The completed server.py:

```
import flask
import sqlite3
from flask import request

app = flask.Flask(__name__)

@app.route('/tutorial', methods=['GET'])
def tutorial():
    name = request.args.get('name', '')
    con = sqlite3.connect('my-db.db')

    if len(name) > 0:
        # insert into db
        insertQuery = "INSERT INTO STUDENT (NAME) values (?);"
        con.execute(insertQuery, (name,))
        con.commit()

    # select from db
    cursor = con.execute("SELECT NAME from STUDENT;")

    students = []
    for row in cursor:
        students.append(row[0])

    con.close()

    outdata = {
        "course_code": "COMP3330",
        "course_name": "Interactive Mobile Application Design and Programming",
        "teachers": ["Dr. T.W. Chim", "Mr. C.K. Lai", "Mr. X. Wang"],
        "students": students
    }
    return outdata


# adds host="0.0.0.0" to make the server publicly available
app.run(host="0.0.0.0")
```