**Webex Teams Hackathon 2018**



**Lab2 – Using SQLite to Store Data in Python REST API Back End**

# Objectives

In this lab, you will complete the following objectives:

* import sqlite3 module
* create database and table
* store data in the database
* get data from the database

# Background / Scenario

Flask is only able to server one client at a time, therefore in a production environment runs multithreaded, meaning multiple instances of the app are running in parallel. This can result in unexpected outcomes when using global variables to store data. Practically the client is served each time by a randomly selected instance of the app, making the data inconsistent.

To solve this issue, it is a good practice to store all our global data of the app in a database. SQLite is a lightweight database system which stores data in a single file. The SQLite library is already included in your Python installation, and it is easy-to-use.

When completed, the **lab2-sqlite.py** program will store all the global values of the previous Lab program in an SQLite database.

# Required Resources

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

**Step 1: Import sqlite3 module**

In this step, you will create your next project based on the Lab1 project.

1. Create your project directory called lab2–sqlite and copy the final version of your lab2-sqlite.py from lab1-back-end-python directory. Rename the new file to lab2-sqlite.py.
2. Edit your lab2-sqlite.py file and import the SQLite module. Add the following text to the beginning of the file:   
   import sqlite3

**Step 2: Create database and table**

1. Create a new function called initDatabase
2. Give the command which will create a new database file if it not exists and create a connection object:

conn = sqlite3.connect('about.db')

With this command the about.db file will be created in the root folder of your app.

1. Create a cursor for the database connection. Cursor object can be used to execute SQL commands.

cur = conn.cursor()

1. Execute an SQL query using the ‘execute’ method of the previously created cursor object. The query will create a person table with three fields:

cur.execute("CREATE TABLE IF NOT EXISTS person (id INTEGER PRIMARY KEY, name VARCHAR(100), age INTEGER)")

1. To save (commit) the changes, use the commit method of the connection object:

con.commit()

1. Call the initDatabase function. Insert the following text right before the app.run() command:

initDatabase()

1. Start your app. In the command line type python lab2-sqlite.py
2. Check the file system. A new about.db file should appear in the root folder of your project.
3. Stop your back end app. Press Crtl-C in the command windows.

**Step 3: Create functions for storing and fetching data to/from the database**

1. Create a function named fetchDataFromDatabase. It will fetch the last record from the person table.

def fetchDataFromDatabase():

with sqlite3.connect('about.db') as conn:

cur = conn.cursor()

result = cur.execute("SELECT \* FROM person ORDER BY id DESC;").fetchone()

return jsonify(id = result[0], name = result[1], age = result[2])

**Note:** result will contain the fetched data in a tuple. A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets. E.g.: (2, “Charles Webex”, 15). We convert the result into JSON format using the jsonify command.

1. Create a function named pushDataToDatabase. It will insert a new record to the person table using values of name and age arguments:

def pushDataToDatabase(name, age):

with sqlite3.connect('about.db') as conn:

cur = conn.cursor()

sql = f"INSERT INTO person (name, age) VALUES ('{name}', {age});"

cur.execute(sql)

conn.commit()

**Note:** You can embed a value stored in a variable into a string if you put an extra ‘f’ letter at the beginning of the string and you insert the variable name in curly brackets in the appropriate position.

1. Add the first record to the table. Insert the following line right after the initDatabase() command:

pushDataToDatabase("Charles Webex", 15)

1. Modify /api/about GET and POST endpoints to use the database functions above.

@app.route("/api/about", methods = ['POST', 'GET'])

def about():

if request.method == 'GET':

return fetchDataFromDatabase()

elif request.method == 'POST':

r = request.json

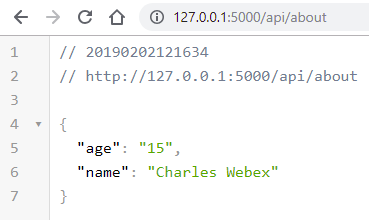
name = r["name"]

age = r["age"]

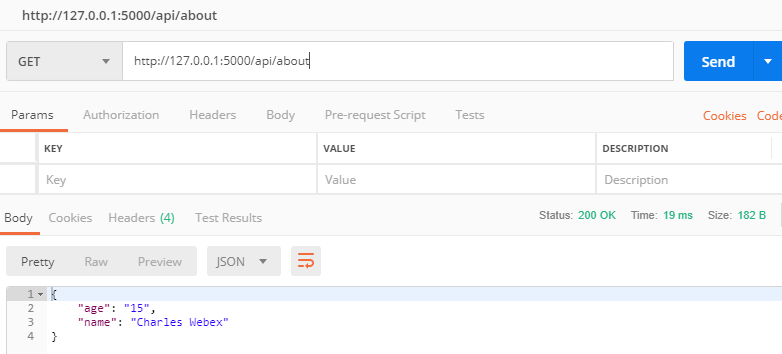
pushDataToDatabase(name, age)

return jsonify(name = name, age = age)

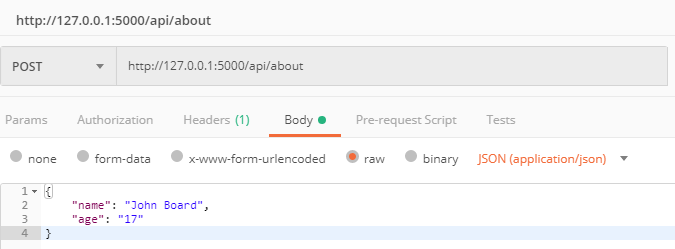
1. Launch your back end app. In the command line type python lab2-sqlite.py
2. Open your Chrome browser and type the following url: <http://127.0.0.1:5000/api/about>



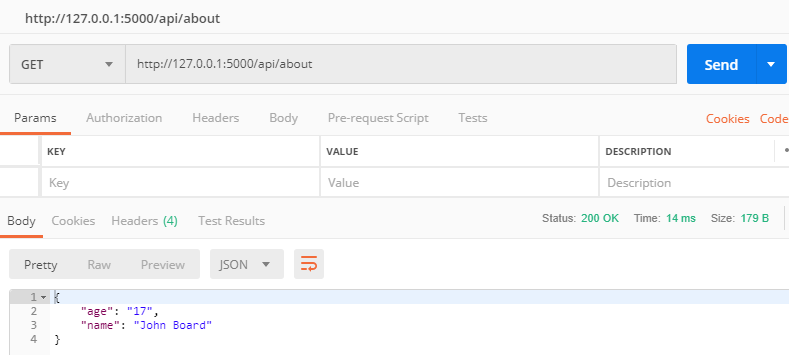
1. Test your GET endpoint in Postman:



1. Add a new record with your POST endpoint in Postman:



1. Check the new state using your GET endpoint.



1. Stop your back end app. Press Crtl-C in the command window.