Flask - API Task

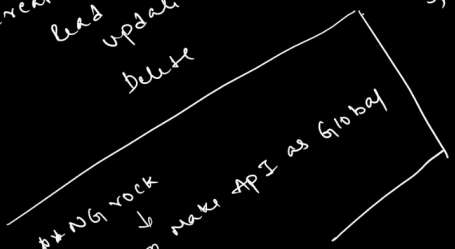
Flask + API

Create APIs for:

1) Creating Table
2) Inserting data into table
3) Update data in table
4) bulk inserting data into table
5) Delete from table

→ MySQL
→ Mongodb
→ Cassandra

CRUD
Create
Read
Update
Delete



6) → for download data from the table.
(select * from)

6 x 3 = 18 APIs.

@app.route

Json format

table : tablename
col1 : text
col2 : col2
ID : text ID
pwd : password.

we can use 'nested JSON' to give col data types

we can also use:

@app.route ('/table/create')
@app.route ('/table/update')

Databases - APIs

Flask - API Task

1) Giving the credentials in .py file & using them in our flask-API code.

```
# creds.py
"""
This  contains the credentials for the MySQL database."""
sqlhost = 'localhost'
sqluser = '[redacted]'
sqlpassword = '[redacted]'
```

```
# credstest.py
import mysql
from flask import Flask, render_template, request, jsonify

import creds

# install mysql-connector-python
import mysql.connector as connection
from mysql.connector import DatabaseError

app = Flask(__name__)

"""Here we are connecting to the database but getting credentials from other python file (creds.py)
instead of getting from the postman (to make sure that no one can see our creds in the postman)"""


@app.route('/mysql/testconnection', methods=['POST'])
def db_connection():
    if (request.method == 'POST'):
        mydb = connection.connect(host=creds.sqlhost, user=creds.sqluser, passwd=creds.sqlpassword, use_pure=True)
    res = mydb.is_connected()
    result = 'The connection to MySQL database is established: ' + str(res)
    return jsonify(result)


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

(2) Creating a class & then using it to connect to db by getting the host, user, PD, dbname from postman.

```
# databaseclass.py
class sql():
    def __init__(self, host, user, password):
        self.host = host
        self.user = user
        self.password = password
        self.mydb = connection.connect(host=self.host, user=self.user, passwd=self.password, use_pure=True)
        self.cur = self.mydb.cursor()
    def db_connection(self, dbname):
        self.dbname = dbname
        query = f"Create database {self.dbname}"
        self.cur.execute(query)
        print(f"Database '{self.dbname}' is created")
app = Flask(__name__)
@app.route('/mysql/oops_createdb/', methods=['POST'])
def db_connect():
    if request.method == 'POST':
        hostn = request.json['hostname']
        userr = request.json['userID']
        password = request.json['pwd']
        dbn = request.json['dbn']
        try:
            sqlcreate = sql(hostn, userr, password)
            sqlcreate.db_connection(dbn)

            return jsonify(f"Database {dbn} is created")
        except Exception as err:
            return jsonify('Error: ' + str(err))
if __name__ == '__main__':
    app.run(debug=True)
```

Database:MySQL-APIs / mysql_oops_createdb

POST http://127.0.0.1:5000/mysql/oops_createdb/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hostname": "localhost",
    "userID": "[redacted]",
    "pwd": "[redacted]",
    "dbn": "sql7894"
}
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
"Error: 1007 (HY000): Can't create database 'sql7894'; database exists"
```

(3) Creating a file to write the classes, other file to write the API.

```
# classfile.py
import mysql.connector as connection
class sql():
    def __init__(self, host, user, password):
        self.host = host
        self.user = user
        self.password = password
        self.mydb = connection.connect(host=self.host, user=self.user, passwd=self.password, use_pure=True)
        self.cur = self.mydb.cursor()

    def db_connection(self, dbname):
        self.dbname = dbname
        query = f"Create database {self.dbname}"
        self.cur.execute(query)
        print(f"Database '{self.dbname}' is created")
```

```
# classfile_api.py
import mysql
import csv
import pandas as pd
from flask import Flask, render_template, request, jsonify

# install mysql-connector-python
import mysql.connector as connection
from mysql.connector import DatabaseError
import classfile

app = Flask(__name__)

@app.route('/mysql/oops_createdb2/', methods=['POST'])
def db_connect():
    if request.method == 'POST':
        hostn = request.json['hostname']
        userr = request.json['userID']
        password = request.json['pwd']
        dbn = request.json['dbn']
        try:
            sqlcreate = classfile.sql(hostn, userr, password)
            sqlcreate.db_connection(dbn)
            return jsonify(f"Database {dbn} is created")

        except Exception as err:
            return jsonify('Error: ' + str(err))

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

Database:MySQL-APIs / mysql_oops_createdb2

POST http://127.0.0.1:5000/mysql/oops_createdb2/

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hostname": "localhost",
    "userID": "[redacted]",
    "pwd": "[redacted]",
    "dbn": "sql7894"
}
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
"Error: 1007 (HY000): Can't create database 'sql7894'; database exists"
```

Enter data → Send to server → The server executes & returns the Result. → Result.

④ Here we give all the details in the postman.

(host, user, pwd, dbname). → Server gets those details from the postman, executes & returns the result.

(I had used this approach to design all the APIs for this task)

```python
@app.route('/mysql/dropdb', methods=['POST'])
def drop_db():
    if (request.method == 'POST'):
        hostn = request.json['hostname']
        userr = request.json['userID']
        password = request.json['pwd']
        db = request.json['dbname']

        mydb = connection.connect(host=hostn, user=userr, passwd=password, use_pure=True)
        cur = mydb.cursor()

        try:
            cur.execute(f"drop database {db}")
            # mydb.commit()
            return jsonify(f"Database: {db} is dropped")

        except Exception as err:
            return jsonify('Error: ' + str(err))


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

POST mysql_oops_created! · POST mysql_dropdb

Database:MySQL-APIs / **mysql_dropdb**

POST http://127.0.0.1:5000/mysql/dropdb

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "hostname": "localhost",
    "userID": "[redacted]",
    "pwd": "[redacted]",
    "dbname": "sqldb00"
}
```