RESTful API Project

Custom API Specification

Course: INFO U762 – 01F

Semester: Spring 2024

Instructor: Dr. Grover Walters

Date: April 22, 2024

David C. Miller

**Custom API Specification**

* My custom API will accept API Update requests from the Driver Application to update the following pieces of data in the DB:
  + Temp
  + Weather Description
  + Currency
  + Brewery Name
  + Brewery URL
  + Brewery Phone
* My custom API will accept API Pull requests to see the Travel related data from Los Angeles.
* Python Code Sections:

**API Update Code Section:**

BASE = " http://127.0.0.1:5000/"

APP\_VERSION = "v1/"

data = [{'wtemp': tempF, 'wdescription': CurrentConditionValue, 'currency': USD, 'brewname': BrewName, 'brewurl': BrewURL, 'brewphone': BrewPhone}]

for i in range(len(data)):

response = requests.patch(BASE + APP\_VERSION + "data/" + str(i), data[i])

print(response.json())

input()

response = requests.patch(BASE + APP\_VERSION + "data/2")

print(response.json())

**API Get Code Section:**

BASE = " http://127.0.0.1:5000/"

APP\_VERSION = "v1/"

**Custom API Code:**

#Import Dependencies

from flask import Flask

from flask\_restful import Api, Resource, reqparse, abort, fields, marshal\_with

from flask\_sqlalchemy import SQLAlchemy

#define application and database variables

app = Flask(\_\_name\_\_)

api = Api(app)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db'

db = SQLAlchemy(app)

app\_version = "v1/"

#create the data definition

class DataModel(db.Model):

id = db.Column(db.Integer, primary\_key=True)

wtemp = db.Column(db.Float, nullable=False)

wdescription = db.Column(db.String, nullable=False)

currency = db.Column(db.Float, nullable=False)

brewname = db.Column(db.String, nullable=False)

brewurl = db.Column(db.String, nullable=False)

brewphone = db.Column(db.String, nullable=False)

#outputs to log/screen to verify data visually

#Commented Out. I used the DB Browser from sqlitebrowser.org to confirm that DB looks good.

# def \_\_repr\_\_(self):

# return f"DataModel(wtemp = {wtemp}, wdescription = {wdescription}, currency = {currency}, brewname = {brewname}, )"

#run this statement the first thme to create the database structure

#db.create\_all()

#handle the incoming data request with a parser

#arguments for a put request

data\_put\_args = reqparse.RequestParser()

data\_put\_args.add\_argument("wtemp", type=float, help="Temp in Celsius", required=True)

data\_put\_args.add\_argument("wdescription", type=str, help="Weather Description", required=True)

data\_put\_args.add\_argument("currency", type=float, help="Currency in Euros", required=True)

data\_put\_args.add\_argument("brewname", type=str, help="BrewPub Name", required=True)

data\_put\_args.add\_argument("brewurl", type=str, help="BrewPub URL", required=True)

data\_put\_args.add\_argument("brewphone", type=str, help="BrewPub Phone Number", required=True)

#arguments for an update request

data\_update\_args = reqparse.RequestParser()

data\_update\_args.add\_argument("wtemp", type=float, help="Temp in Celsius", required=True)

data\_update\_args.add\_argument("wdescription", type=str, help="Weather Description", required=True)

data\_update\_args.add\_argument("currency", type=float, help="Currency in Euros", required=True)

data\_update\_args.add\_argument("brewname", type=str, help="BrewPub Name", required=True)

data\_update\_args.add\_argument("brewurl", type=str, help="BrewPub URL", required=True)

data\_update\_args.add\_argument("brewphone", type=str, help="BrewPub Phone Number", required=True)

#Map the types to columns extracted from the database object

resource\_fields = {

'id': fields.Integer,

'wtemp': fields.Float,

'wdescription': fields.String,

'currency': fields.Float,

'brewname': fields.String,

'brewurl': fields.String,

'brewphone': fields.String

}

#Set up the Resource Functions for CRUD

class Data(Resource):

#GET (READ in CRUD)

#@marshal\_with serializes output from the DB as a dictionary (json object) so we can work with it in python

@marshal\_with(resource\_fields)

def get(self, data\_id):

result = DataModel.query.filter\_by(id=data\_id).first()

if not result:

abort(404, message="Could not find data with that id")

return result

#POST (CREATE in CRUD)

@marshal\_with(resource\_fields)

def put(self, data\_id):

args = data\_put\_args.parse\_args()

result = DataModel.query.filter\_by(id=data\_id).first()

if result:

abort(409, message="Data id taken...")

data = DataModel(id=data\_id, wtemp=args['wtemp'], wdescription=args['wdescription'], currency=args['currency'], brewname=args['brewname'], brewurl=args['brewurl'], brewphone=args['brewphone'])

db.session.add(data)

db.session.commit()

return data, 201

#PUT (UPDATE in CRUD)

@marshal\_with(resource\_fields)

def patch(self, data\_id):

args = data\_update\_args.parse\_args()

result = DataModel.query.filter\_by(id=data\_id).first()

if not result:

abort(404, message="Data doesn't exist, cannot update")

if args['wtemp']:

result.wtemp = args['wtemp']

if args['wdescription']:

result.wdescription = args['wdescription']

if args['currency']:

result.currency = args['currency']

if args['brewname']:

result.brewname = args['brewname']

if args['brewurl']:

result.brewurl = args['brewurl']

if args['brewphone']:

result.brewphone = args['brewphone']

db.session.commit()

return result, 200

#DELETE (DELETE in CRUD)

def delete(self, data\_id):

abort\_if\_data\_id\_doesnt\_exist(data\_id)

del Data[data\_id]

return '', 204

#Register the Resource called video to the API (remember to change versions when making changes for submission)

api.add\_resource(Data, "/" + app\_version + "data/<int:data\_id>")

#Run the API body

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)