Task**:**

**Create** simple **web app** in **python** that works with **sql** and **store drink items in SQL table.**

So far implemented the **server side** that implements **Web Server** side while the **Web Client** side is simulated by **Postman**

Web Server that will get REST API REQUESTS (**GET** / **GET** by **id** / **POST** / **DELETE**) from Web Client (**Postman**) and update the table drinks in my DB (**drinks**.**db**)

Task works with packages:

1. **Flask**
2. **Flask\_sqlalchemy**

Flask, **by default** uses **local host ip** address: [**http://127.0.0.1**](http://127.0.0.1)**,** means that only my pc can access this URL (IP address)

**SQLalchemy**, **by default** gives a table name = name of a class that defines it’s db.Model

If I defined that the class that represents a drink item that I store in table ***drinks*** in my DB ***drinks.db*** has name **Drink**

A screen shot of a computer program

AI-generated content may be incorrect.

Then the table that will be created – will have default name - **drink** (=same name as a class but lover case !!!)

**But** we can define table name explicitly by adding attribute to the class:

**\_\_tablename\_\_= ‘my\_drinks’** # see how I did in my class

What is the app context:

This is all that you wish to happen must be written in the context of the app.

**Due to the fact we use Flask, the main is not a context of the app !!!** I mean

if \_\_name\_\_ == ‘\_\_main\_\_’:

xxxxxx

xxxxx

xxx

all the xxxx are not in the context of the app therefore will not run – main will not run if we run app with Flask.

What we must define at the very beginning of the run of the app:

* Our DB will be based on the package SQLAlchemy

1. # 1 Create db obj - of the type SQLAlchemy (no word about the app)  
   db = SQLAlchemy()

Our app is created by package Flask

1. # 2 Create app (use Flask module)  
   app = Flask(\_\_name\_\_)

Connection between application and the db

1. # 3 Here we connect between DB and the app  
   # Configure the app to work with SQLALCHEMY (app will use db called: drinks.db)  
   app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///drinks.db'  
   app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = *False*

Initialization of the db by the application

1. # 4 Initialize db with the app - this is actual binding between app and db  
   db.init\_app(app)

Simply storing the path of the db (**drinks.db** will be created in this path)

1. # db\_path is set with a location where will be created a db in the project  
   db\_path = "instance/drinks.db"

Delete the **drinks.db** at each execution of the application

Lets look on the class **Drink**:

1. # 5 create data base model - in simple words it creates a table of the type Drink, which name = class name but in lowercase  
   # class defines the class model of the table.  
   # DB is drinks.db but it will contain 1 table inside it, called drink  
   # because we use SQLAlchemy, SQLAlchemy automatically creates a table named drink in the database.  
   # if we use this attribute in the class Drink: \_\_tablename\_\_ = 'my\_drinks' # Set the table name explicitly to be 'my\_drinks' instead of drink (table name = DB name)  
   *class* Drink(db.Model):  
    # I set here the explicit name for the table that will be created in my DB  
    # If we do not set the name explicitly, its default name will be as a class name, but in lover case (e.g: drink)  
    \_\_tablename\_\_ = 'my\_drinks'  
     
    # here I define that in table 'drinks' will be 3 columns: id, name, description  
    id = db.Column(db.Integer, primary\_key=*True*)  
    name = db.Column(db.String(80), unique=*True*,nullable=*False*)  
    description = db.Column(db.String(80))  
     
    # magic func of the class for object (self) representation  
    *def \_\_repr\_\_*(self):  
    *return* f"{self.name}-{self.description}"

Here we created the type of the each raw we will put in the table

We gona put Drink items in the table

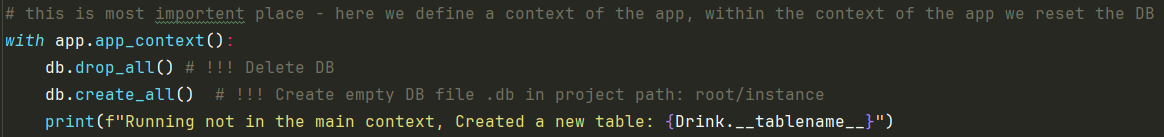
Each drink item has:

1. Id
2. Name
3. Description
4. Each time we run the app it should start from the empty table so we should delete old DB and the table
5. # remove db from the project if it created previously  
   *if* os.path.exists(db\_path):  
    os.remove(db\_path)  
    print("Old database deleted.")

Deletion must be done after we define the DB and connected between: **application** & **db** & **class model**

1. **The most important thing is to create a new DB – new context (as idea)**

**This must be done after all is ready and known**



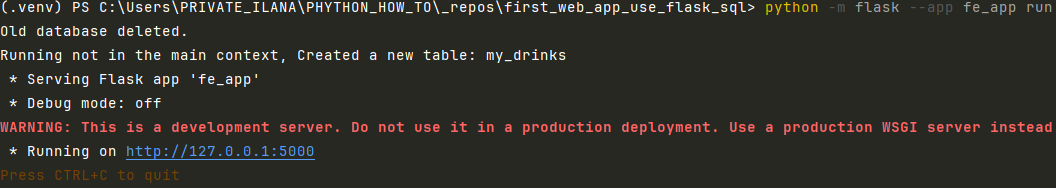
Now is the time to define hitters for our app:

These are functions that will be invoked upon requests from the WEB Client side (that I currently do from **Postman**)

To run the application via Flask in Pycharm, do: \* my application is associated today only with a file: **fe\_app.py**

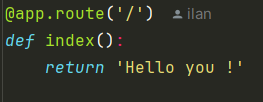
**python -m flask --app fe\_app run**

when we run the application we see on the command line this:



If we click on the ip in Pycharm will be opened a browser  
with: “'Hello you !' ”

This is because the first hitter func:

  
The decorator **@app.route** is written above each hitter. Flask binds between hitter func and the URL

**route** method receives 2 params:

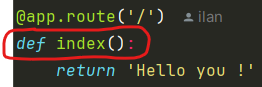
**rule** (=**URL**)

**option** (here we can write: **method**=[**DELETE**] or **method**=[**POST**])

A graph on a white background

AI-generated content may be incorrect.**Or** if from the Postman send request of the type **GET: http://127.0.0.1:5000**

Will be called hitter:



And will be printed “'Hello you !' ”

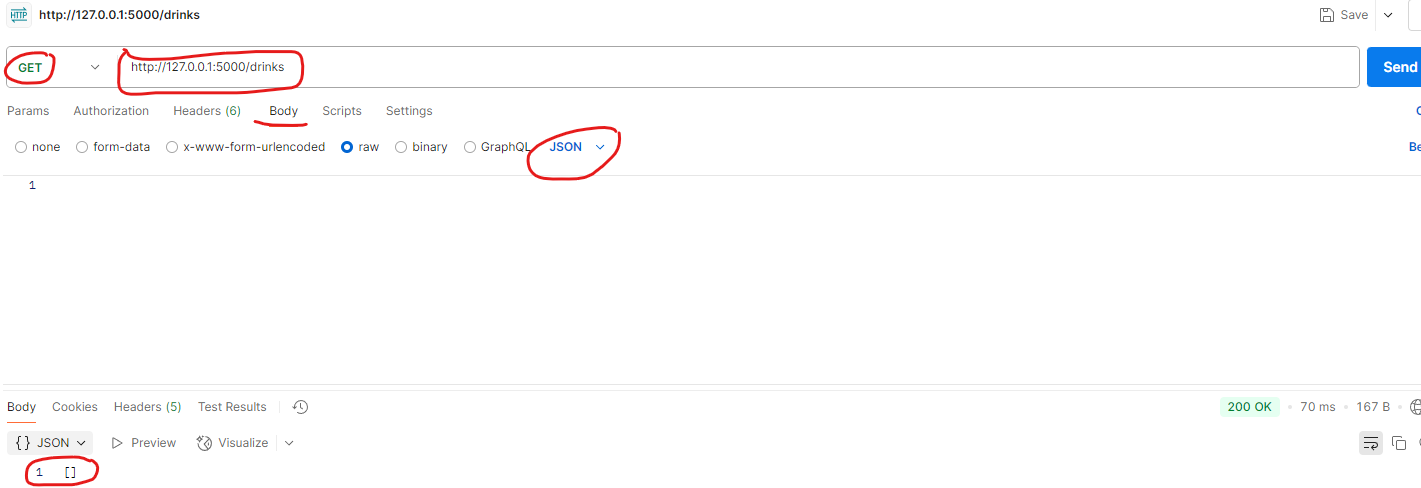
This hitter func is called:

A screen shot of a computer program

AI-generated content may be incorrect.

Pay attention: it is GET request that asks to get all that is under the URL 127.0.0.1/5000

To hit any other URL you need a new hitter func, for example if we need to get all the drink from a table drinks the URL (the route) is: GET [**http://127.0.0.1:5000/drinks**](http://127.0.0.1:5000/drinks)

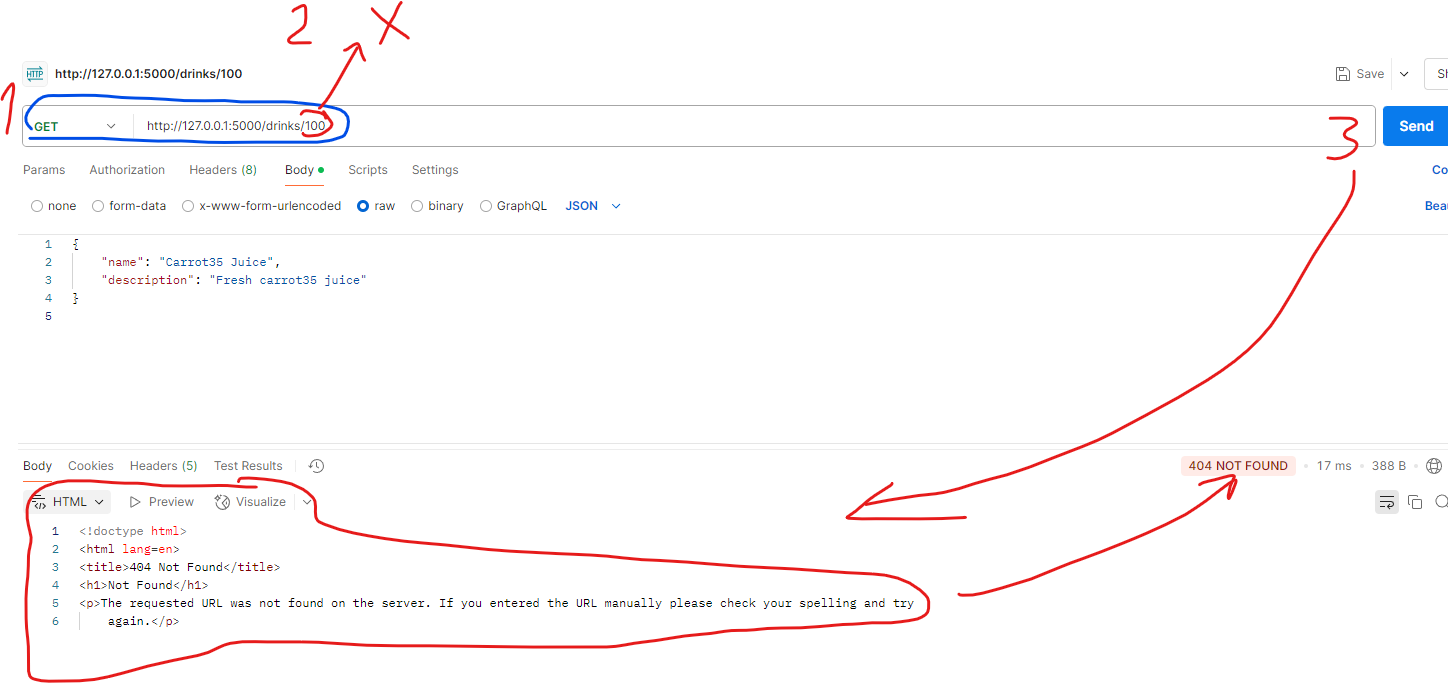
In case it is empty we should get []

In case we wish to insert new item into the table we use POST

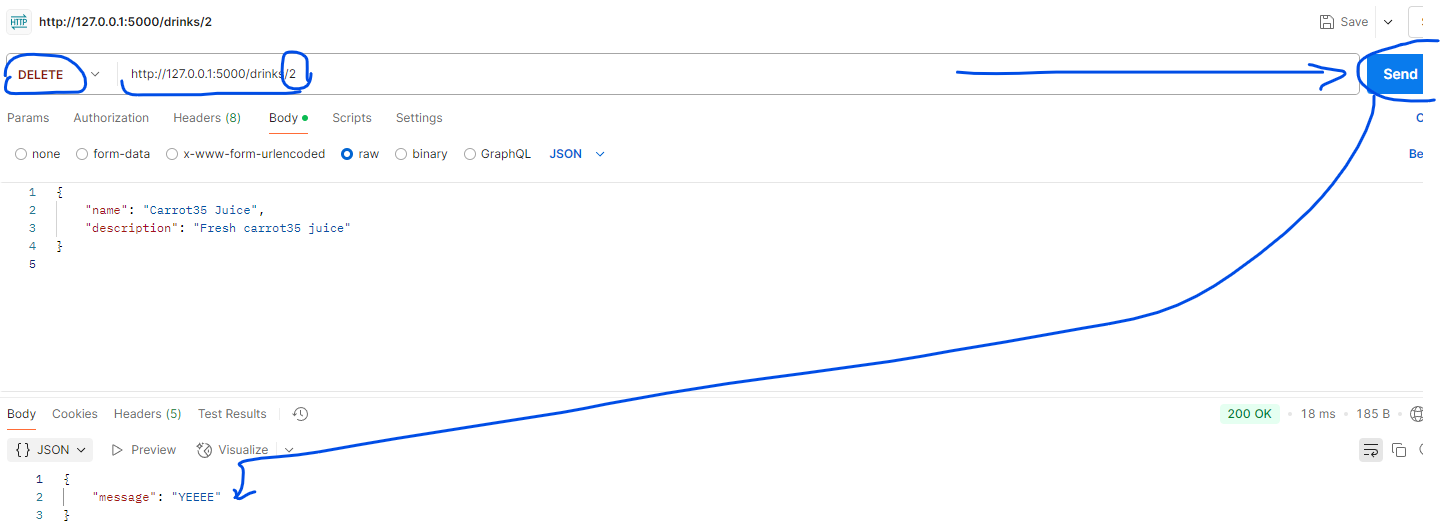
A screenshot of a white board with red writing

AI-generated content may be incorrect.To get Drink item by id, use request get with id:

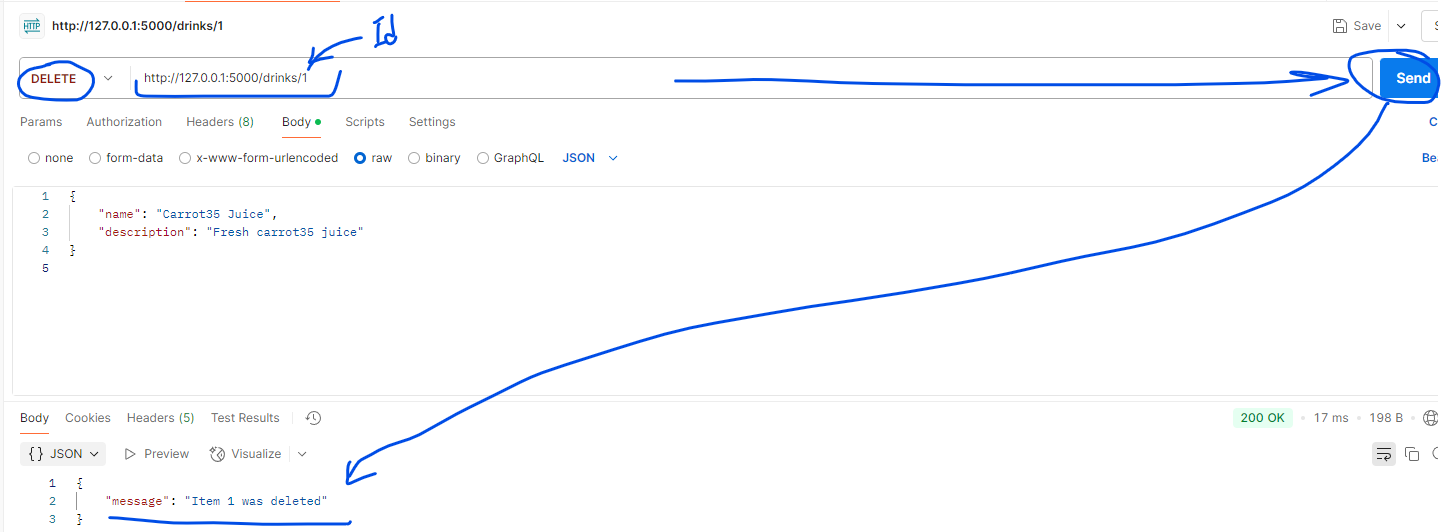
[**http://127.0.0.1:5000/drinks/1**](http://127.0.0.1:5000/drinks/1)

if such id isn’t existing, you will get 404 Client Error

To delete Drink item by id use request: DELETE

to delete item from the table I need REST API of the type DELETE with index

DELETE: URL/index

**DELETE:** [**http://127.0.0.1:500/drinks/id**](http://127.0.0.1:500/drinks/id)

Commands that we use to operate Server with Postman that we sow so far are:

|  |  |  |
| --- | --- | --- |
| **From where we run this cmd** | **Cmd** | **Meaning of the cmd** |
| cli, Pycharm | python -m flask --app fe\_app run | **Run the Application** by using Flask package |
| Postman | http://127.0.0.1:5000 | **GET:**  To see the URL is accessible + to get: “'Hello you !' ” |
| Also possible from cli | Click in cli of Pycharm on the url:  <http://127.0.0.1/5000> | **GET:**  Will be opened a **browser** and you will see the message |
| Postman | http://127.0.0.1:5000/drinks | **GET:**  **Retrieve** **all existing drinks**.  If no drinks were posted you should get in Postman this output:  **[]**  \* Same url you can write in browser and get same output. |
| Postman | http://127.0.0.1:5000/drinks/1 | **GET:**  **Retrieve** specific  **drink** (by drink index = 1)  If such drink isnt’ existing, you should get ERROR |
| Postman | <http://127.0.0.1:5000/drinks/>  Exp:  In the body write:  {      "name": "Carrot35 Juice",      "description": "Fresh carrot35 juice"  } | **POST:**  **Add** new **drink** to the **drinks table** |
| Postman | <http://127.0.0.1:5000/drinks/>1  output should be: | **DELETE:**  **Delete** a **drink** from the **drinks table** |