**from flask import Flask, request**

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

**stores\_list = [**

**{**

**"name" : "Demo Store"**

**, "items" : [**

**{**

**"name" : "chair"**

**, "price" : 15.99**

**}**

**]**

**}**

**]**

**@app.get( '/' ) # http://127.0.0.1:5000/**

**def welcomeNote():**

**return { "Welcome" : "Welcome to Our Grocery App" }**

**@app.get( '/stores' ) # http://127.0.0.1:5000/stores**

**def get\_stores():**

**return { "Stores" : stores\_list }**

**# return str( stores )**

**@app.post( '/addStore' ) # http://127.0.0.1:5000/stores**

**def create\_store():**

**request\_data = request.get\_json()**

**print( f"request\_data = { request\_data }" )**

**new\_store = {"name" : request\_data["name"], "items" : [] }**

**stores\_list.append( new\_store )**

**return new\_store, 201**

**@app.get( '/stores/<string:store>' ) # http://127.0.0.1:5000/store/store\_name**

**def getStore(store):**

**try:**

**selected\_store = list( filter( lambda i : i['name'] == store , stores\_list ) )[0]**

**return {"selected\_store" : selected\_store }, 200**

**except Exception as e:**

**return { "Error" : f"No Data Found, Exception = {e}" }, 404**

**@app.post( '/stores/<string:store>/addItem' ) # http://127.0.0.1:5000/store/store\_name/addItem**

**def addItem(store):**

**request\_data = request.get\_json()**

**item\_data = { "name" : request\_data["name"]**

**, "price" : request\_data["price"]**

**}**

**try:**

**selected\_store = list( filter( lambda i : i['name'] == store , stores\_list ) )**

**print( f"selected\_store = { selected\_store }" )**

**selected\_store = selected\_store[0]**

**selected\_store["items"].append( item\_data )**

**return { "selected\_store" : selected\_store }, 201**

**except Exception as e:**

**return { "Error" : f"No Data Found, Exception = {e}" }, 404**

**# app.run()**

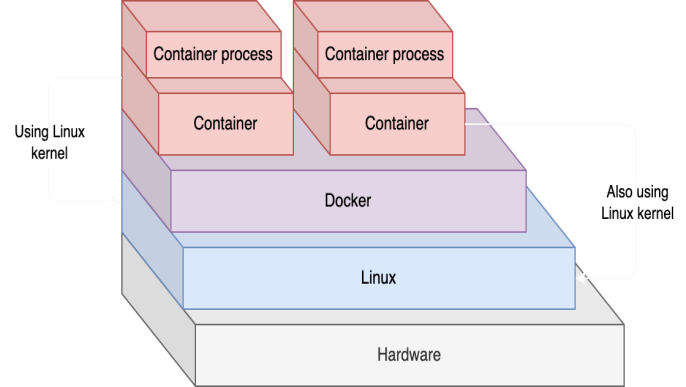
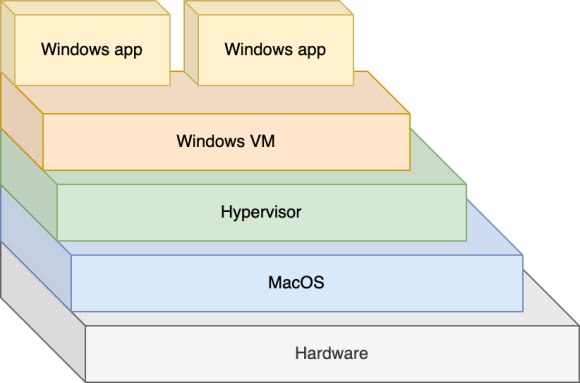
**###################### Docker Start #####################**

**# What is Docker**

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

**# Why Docker**

With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker’s methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.



**# What is Docker Container ?**  
A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine.

Let’s imagine you have written some code and want to run it on your friend’s PC. If you want to run on your friend’s PC then, you need to install dependencies and required libraries as well for smoother run.  
  
Docker runs on Linux machine properly as it uses linux kernal. So If you are having Mac/ Windows then, you need a hypervisor where you can run linux OS and then, you can use Docker.



**# Can you run an Ubuntu image when the host is Linux but not Ubuntu ?**

Since the Linux kernel is the same between distributions, and since Docker containers only use the host's kernel, it doesn't matter which distribution you are running as a host. You can run containers of any distribution with any other distribution as a host.

**# What is Docker Image** **?**  
 If you want to create docker container in order to run container process( Software/ Code ) then, you need to create image document first which will have all the steps and tell docker how to proceed for docker container creation. You need to build this document/ image for docker container creation. You just have to build your code and this image document on your friend’s system for code run.



And then, you can run your container to run program or docker container.

**# Docker Image Document :-**

FROM python:3.10 # You are saying Docker to install # python 3.10 version

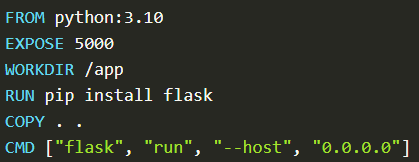
EXPOSE 5000 # You are telling docker to run the # program on port 5000

WORKDIR /app # You are giving docker your code path # i.e., app.py path

RUN pip install flask # Your are telling docker to install # flask using pip

COPY . . # Copy <FOLDER\_DIR> <FILE\_DIR>

CMD ["flask", "run", "--host", "0.0.0.0"] # Docker will run # flask program on # host 0.0.0.0



**IMPORTANT NOTE : The name of Docker files doesn't have any extension. It's just Dockerfile with capital D and lowercase f.**

**# How to run a Docker container ?**

**Step 1 : Install Docker Desktop**

Docker Desktop is an application to help you manage your images and containers.

Download it and install it here:

<https://www.docker.com/products/docker-desktop/.>

**Step 2 : Create your Docker image**

**Step 2.1 :-** Open a terminal

docker build -t rest-apis-flask-python .

When the command is finished, you should see this (among other things):

**=> [2/4] WORKDIR /app 0.4s  
 => [3/4] RUN pip install flask 2.9s**

**=> [4/4] COPY . . 0.0s**

**=> exporting to image 0.1s**

**=> exporting layers 0.1s**

**=> writing image sha256:d9a68a03f868e74bca48567dfc9a0b702d1618941a71b77de12ff14e908ba155 0.0s**

**=> naming to docker.io/library/rest-apis-first-rest-api 0.0s**

IF you get Error while running : docker build -t rest-apis-flask-python .  
  
 then, open terminal and try this :

"C:\Program Files\Docker\Docker\DockerCli.exe" -SwitchDaemon

You should be able to see following output,



If build is successful then, run following command.

docker run -d -p 5000:5000 rest-apis-flask-python

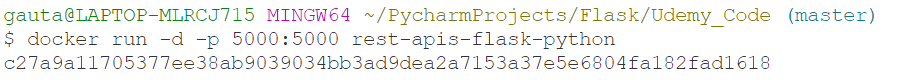
1) -d runs the container in the background, so that you can close the terminal and the container keeps running.

2) -p 5000:5000 maps port 5000 in your computer to port 5000 in the container.

3) rest-apis-flask-python is the image tag that you want to run.

You should see something like this as your output:

9f3c564ac64a1723069dda0e80becb70d3697d4bfcbcb626cd5add0c65df173f



Or Else you can go Docker Desktop and then, Run the Image as container for given PORT, host, etc.

That's the ID of the container. **If you're not using Docker Desktop, you need this ID in order to stop the container** later (with docker rm 9f3c564, that's the first few characters of the ID).

**############################ Docker End #####################**

**Let’s get back to out flask Code.**

Write in order to track any store. We are using store name.  
Let’s imagine if we have 2 stores having same name and we want to add items in 1st store then, python may get confuse between the duplicate store We Should be using unqiue\_id to identify any particular store.

Before Adding Store.  
  
  
Adding item Store 1 in stores but the problem is that we have 2 stores with same name and we need to add item in 2nd Store 1.

We have following item dictionary which we want to add.

{

                "name": "Table",

                "price": 25

            }

As you can see that item got added to 1st “Store 1” not the 2nd “Store 1”.



To avoid these kind of problems, we need to a unique identifier for each store.

For this we need do little changes in our code.

First of all, we need to make 3 following files to make our code much dynamic and optimized,

1. Make requirements.txt where you have to mention all the dependent libraries.

flask

falsk-smorest

python-dotenv

1. Make .flaskenv file where you can define your variable for scalability and change the parameter dynamically.

FLASK\_APP=app

FLASK\_ENV=development

1. Make db.py which will have stores and item dictionary. We are going to remove stores list from app.py now.

stores = {}I  
 items = {}

We are going to store data in stores and items like following :-

stores = {

    "17237446a67f4ed9ab51c8612fd3aa8e": { # store\_id1

        "items": [],

        "store\_id": "17237446a67f4ed9ab51c8612fd3aa8e",

        "store\_name": "Store 1"

    },

    "5fde263d00aa4519af306f1ff360a4ff": { # store\_id2

        "items": [],

        "store\_id": "5fde263d00aa4519af306f1ff360a4ff",

        "store\_name": "Store 2"

    }

}

items = {

    "7c648b638fc14df5ad98e4520cb16346": { #item\_id1

        "item\_id": "7c648b638fc14df5ad98e4520cb16346",

        "item\_name": "Chair",

        "item\_price": 20,

        "store\_id": "5fde263d00aa4519af306f1ff360a4ff" # store\_id2

    },

    "8c50fc4da6b1476d9446ece6cd5efbf5": {

        "item\_id": "8c50fc4da6b1476d9446ece6cd5efbf5", #item\_id1

        "item\_name": "table",

        "item\_price": 21,

        "store\_id": "5fde263d00aa4519af306f1ff360a4ff" # store\_id2

    }

}

Also, we will use from flask\_smorest import abort to implement aborting instead of return startement. Abort terminates the API request during runtime request goes wrong and it does auto-return

So Our New Python code will be following :-

**######### Code - Version 1/ V1 Start #############**

import uuid

from flask import Flask, request

from db import \*

from flask\_smorest import abort

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

@app.get( '/' ) # http://127.0.0.1:5000/

def welcomeNote():

    return { "Welcome" : "Welcome to Our Grocery App" }, 200

@app.post( '/addStore' ) # http://127.0.0.1:5000/addStore

def create\_store():

    store\_data = request.get\_json()

    store\_id = uuid.uuid4().hex

    store\_data['store\_id'] = store\_id

    store\_data['items'] = []

    print( f"store\_data = { store\_data }" )

    stores[store\_id] = store\_data

    return store\_data, 201

@app.get( '/stores' ) # http://127.0.0.1:5000/stores

def getallstores():

    return stores, 200

@app.get( '/store/<string:store\_id>' ) # http://127.0.0.1:5000/store/store\_id

def getStore(store\_id):

    try:

        return stores[store\_id], 200

    except KeyError as e:

        abort( 404, message= f"Store not Found, Exception = {e}" )

@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( item\_data["store\_id"] not in stores ):

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

@app.get( '/items' ) # http://127.0.0.1:5000/items

def getallitems():

    return items, 200

@app.get( '/item/<string:item\_id>' ) # http://127.0.0.1:5000/item/item\_id

def getitem(item\_id):

    try:

        return items[item\_id], 200

    except KeyError as e:

        abort( 404, message= f"Item not Found, Exception = {e}" )

# app.run()

**######### Code - Version 1/ V1 End #############**

################## Validating functionalities Start ####################

1. Making sure that we are passing enough number of parameter while calling API

**# Existing Code :-**  
@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( item\_data["store\_id"] not in stores ):

        # return { "Error" : f"Store not found having store\_id : { item\_data['store\_id'] }" }

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

**# New Code :-**

@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( ("store\_id" not in item\_data) # New Added Code

        and ("item\_name" not in item\_data) # New Added Code

        and ("item\_price" not in item\_data) ): # New Added Code

        abort( 404, message= f"API does not have required parameters" ) # New Added Code

    if( item\_data["store\_id"] not in stores ):

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

1. Making sure that we don’t add duplicate item for the same store.

**# Existing Code :-**  
@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( ("store\_id" not in item\_data) # New Added Code

        and ("item\_name" not in item\_data) # New Added Code

        and ("item\_price" not in item\_data) ): # New Added Code

        abort( 404, message= f"API does not have required parameters" ) # New Added Code

    if( item\_data["store\_id"] not in stores ):

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

**# New Code :-**

@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( ("store\_id" not in item\_data)

        and ("item\_name" not in item\_data)

        and ("item\_price" not in item\_data) ):

        abort( 404, message= f"API does not have required parameters" )

    if( item\_data["store\_id"] not in stores ):

        # return { "Error" : f"Store not found having store\_id : { item\_data['store\_id'] }" }

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    for each\_item in items.values(): # New Added Code

        if( ( each\_item["store\_id"] == item\_data["store\_id"] ) # New Added Code

        and ( each\_item["item\_name"] == item\_data["item\_name"] ) ): # New Added Code

            abort( 404, message= f"Duplicate Item present in store\_id : # New Added Code { item\_data['store\_id'] }" ) # New Added Code

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

################## Validating functionalities End ####################

Let’s do similar validation in **/addStore** api also

Making sure that we are passing enough number of parameter while calling API

**# Existing Code :-**

@app.post( '/addStore' ) # http://127.0.0.1:5000/addStore

def create\_store():

    store\_data = request.get\_json()

    store\_id = uuid.uuid4().hex

    store\_data['store\_id'] = store\_id

    store\_data['items'] = []

    print( f"store\_data = { store\_data }" )

    stores[store\_id] = store\_data

    return store\_data, 201

**# New Code :-**

@app.post( '/addStore' ) # http://127.0.0.1:5000/addStore

def create\_store():

    store\_data = request.get\_json()

    if( "store\_name" not in store\_data ):

        abort( 404, message= f"API does not have required parameters" )

    store\_id = uuid.uuid4().hex

    store\_data['store\_id'] = store\_id

    store\_data['items'] = []

    print( f"store\_data = { store\_data }" )

    stores[store\_id] = store\_data

    return store\_data, 201

**######### Code - Version 2/ V2 Start #############**

import uuid

from flask import Flask, request

from db import \*

from flask\_smorest import abort

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

@app.get( '/' ) # http://127.0.0.1:5000/

def welcomeNote():

    return { "Welcome" : "Welcome to Our Grocery App" }, 200

@app.post( '/addStore' ) # http://127.0.0.1:5000/addStore

def create\_store():

    store\_data = request.get\_json()

    if( "store\_name" not in store\_data ):

        abort( 404, message= f"API does not have required parameters" )

    store\_id = uuid.uuid4().hex

    store\_data['store\_id'] = store\_id

    store\_data['items'] = []

    print( f"store\_data = { store\_data }" )

    stores[store\_id] = store\_data

    return store\_data, 201

@app.get( '/stores' ) # http://127.0.0.1:5000/stores

def getallstores():

    return stores, 200

@app.get( '/store/<string:store\_id>' ) # http://127.0.0.1:5000/store/store\_id

def getStore(store\_id):

    try:

        return stores[store\_id], 200

    except KeyError as e:

        # return { "Error" : f"Store not Found, Exception = {e}" }, 404

        abort( 404, message= f"Store not Found, Exception = {e}" )

@app.post( '/addItem' ) # http://127.0.0.1:5000/addItem

def addItem():

    item\_data = request.get\_json()

    print( f"item\_data = { item\_data }")

    if( ("store\_id" not in item\_data)

        and ("item\_name" not in item\_data)

        and ("item\_price" not in item\_data) ):

        abort( 404, message= f"API does not have required parameters" )

    if( item\_data["store\_id"] not in stores ):

        # return { "Error" : f"Store not found having store\_id : { item\_data['store\_id'] }" }

        abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

    for each\_item in items.values():

        if( ( each\_item["store\_id"] == item\_data["store\_id"] )

        and ( each\_item["item\_name"] == item\_data["item\_name"] ) ):

            abort( 404, message= f"Duplicate Item present in store\_id : { item\_data['store\_id'] }" )

    item\_id = uuid.uuid4().hex

    item\_data["item\_id"] = item\_id

    items[item\_id] = item\_data

    return item\_data, 201

@app.get( '/items' ) # http://127.0.0.1:5000/items

def getallitems():

    return items, 200

@app.get( '/item/<string:item\_id>' ) # http://127.0.0.1:5000/item/item\_id

def getitem(item\_id):

    try:

        return items[item\_id], 200

    except KeyError as e:

        # return { "Error" : f"Item not Found, Exception = {e}" }, 404

        abort( 404, message= f"Item not Found, Exception = {e}" )

app.run()

**######### Code - Version 2/ V2 End #############**

Now we need to creare APIs for DELETE and Update for Store as well as for Items

**# Code for del item :-**

@app.delete( '/item/<string:item\_id>' ) # http://127.0.0.1:5000/item/item\_id

def deleteitem(item\_id):

    try:

        op = items.pop( item\_id )

        return op, 202

    except KeyError as e:

        abort( 404, message= f"Item not Found having item\_id = { item\_id }" )

**# Code for del store :-**

@app.delete( '/store/<string:store\_id>' ) # http://127.0.0.1:5000/store/store\_id

def deletestore(store\_id):

    try:

        op = stores.pop( store\_id )

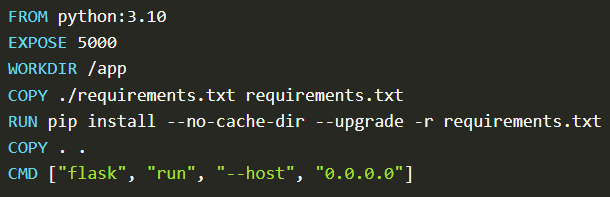
        return op, 202

    except KeyError as e:

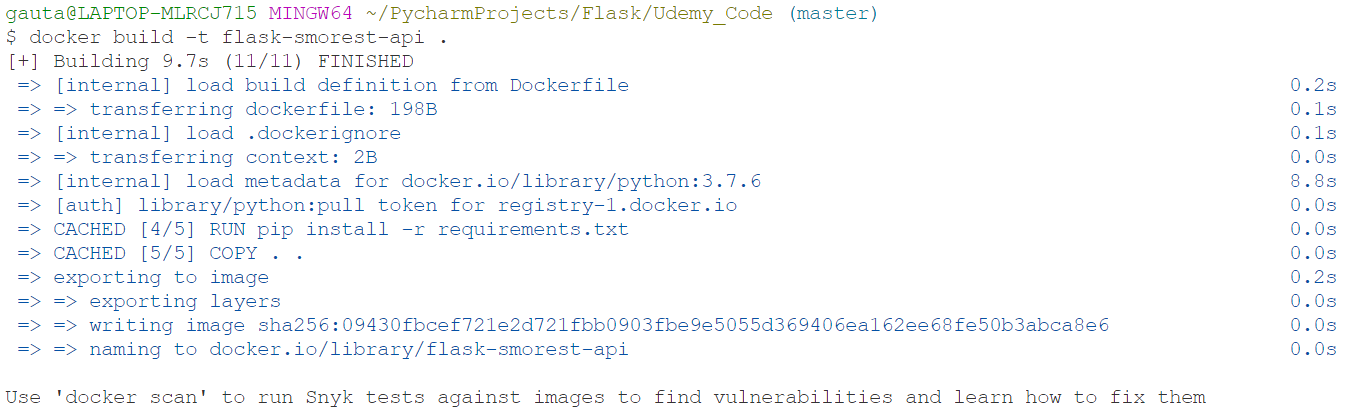
        abort( 404, message= f"Store not Found having store\_id = { store\_id }" )

**###################### Docker Start #####################**

If you want to link your code with docker and run it. So that you don’t have to run the code every time whenever you do changes then, you should create a volumn.



Terminal Command :-



Linux : Now this :-

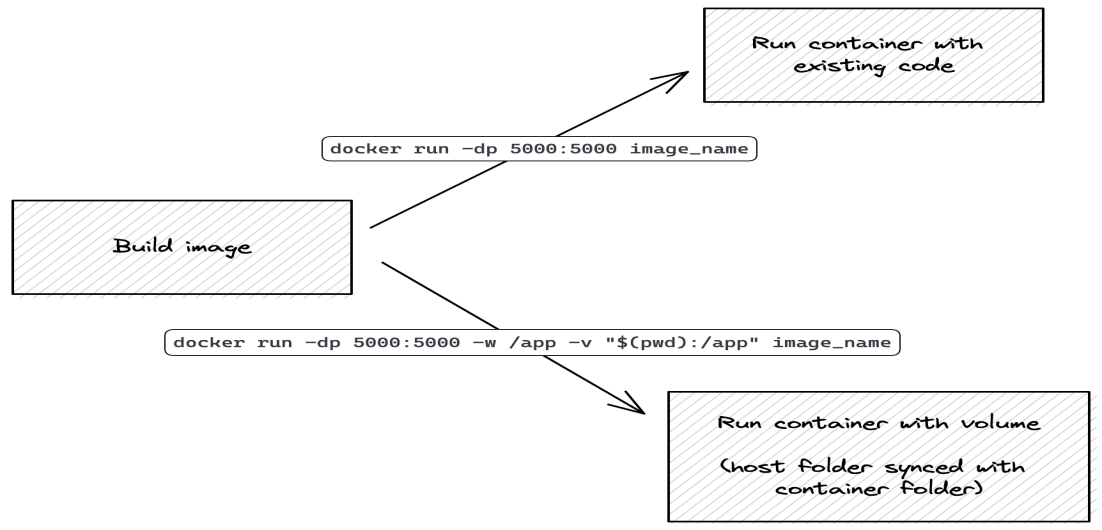
docker run -dp 5000:5000 -w /app -v "$(pwd):/app" flask-smorest-api

Windows : Now this :-

docker run -dp 5000:5000 -w /app -v "/c/Documents/yourproject:/app" flask-smorest-api

docker run -dp 5000:5000 -w /app -v "/c/Users/gauta/PycharmProjects/Flask/Udemy\_Code:/app" flask-smorest-api





**############################ Docker End #####################**

Now, you need to make variable in postman in order to make postman little more flexible for future enhancement.

**Make port = 5000 and url = <http://127.0.0.1:{{port}}>**

After this Level of information. It’s time to improve our flask code in a little bit more better form and structure.

**##########User Blueprints and MethodView in Flask Start##############**

############## app.py Code Start #############

Let’s write store related API code in a separate stores.py file and items related API code in a separate items.py files

from flask import Flask

from flask\_smorest import Api

from resources.item import blp as ItemBlueprint

from resources.store import blp as StoreBlueprint

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

app.config["PROPAGATE\_EXCEPTIONS"] = True

app.config["API\_TITLE"] = "Stores REST API"

app.config["API\_VERSION"] = "v1"

app.config["OPENAPI\_VERSION"] = "3.0.3"

app.config["OPENAPI\_URL\_PREFIX"] = "/"

app.config["OPENAPI\_SWAGGER\_UI\_PATH"] = "/swagger-ui"

app.config["OPENAPI\_SWAGGER\_UI\_URL"] = "https://cdn.jsdelivr.net/npm/swagger-ui-dist/"

api = Api(app)

api.register\_blueprint(ItemBlueprint)

api.register\_blueprint(StoreBlueprint)

############## app.py Code End #############

############## Store.py Code Start #############

import uuid

from flask import request

from flask.views import MethodView

from flask\_smorest import abort, Blueprint

from db import stores

from schemas import StoreSchema

blp = Blueprint( 'stores', \_\_name\_\_, description="Opeations on stores" )

# The Blueprint arguments are the same as the Flask Blueprint1, with an added optional description keyword argument:

# "stores" is the name of the blueprint. This will be shown in the documentation

@blp.route('/store/<string:store\_id>')

class Store( MethodView ):

    def get( self, store\_id ):

        try:

            return stores[store\_id], 200

        except KeyError as e:

            # return { "Error" : f"Store not Found, Exception = {e}" }, 404

            abort( 404, message= f"Store not Found, Exception = {e}" )

    def delete( self, store\_id ):

        try:

            op = stores.pop( store\_id )

            return op, 202

        except KeyError as e:

            abort( 404, message= f"Store not Found having store\_id = { store\_id }" )

    @blp.arguments( "StoreUpdateSchema" )

    def put( self, put\_store\_data, store\_id ):

    # def put( self, store\_id ):

        put\_store\_data = request.get\_json()

        if( not stores.get( store\_id ) ):

            abort( 404, message= f"Item not Found having store\_id = { store\_id }" )

        if( "store\_name" not in put\_store\_data ):

            abort( 404, message= f"API does not have required parameters" )

        stores[ store\_id ]["store\_name"] = put\_store\_data["store\_name"]

        return stores[ store\_id ], 202

@blp.route('/addStore')

class AddStore(MethodView):

    @blp.arguments('StoreSchema')

    def post(self, store\_data): # We need to keep Marshmallow argument at the 2nd place just after self.

                                # as we are implementing Marshmallow technique. So Marshmallow will give the data to argument

                                # in order to validate data using blp.arguments( ItemSchema ) decorator and then, allows

                                # Python to proceed further if everything looks good.

        # store\_data = request.get\_json() # As we are getting item\_data details. So, request.get\_json() is not required

        """

        if( ("store\_name" not in store\_data) and ("store\_type" not in store\_data) ):

            abort( 404, message= f"API does not have required parameters" )

        """

        store\_id = uuid.uuid4().hex

        store\_data['store\_id'] = store\_id

        store\_data['items'] = []

        print( f"store\_data = { store\_data }" )

        stores[store\_id] = store\_data

        return store\_data, 201

@blp.route("/stores")

class Stores( MethodView ):

    def get( self ):

        return stores, 200

############## Store.py Code End #############

############## Item.py Code Start #############

import uuid

from flask import request

from flask.views import MethodView

from flask\_smorest import abort, Blueprint

from db import items, stores

from schemas import ItemSchema, ItemUpdateSchema

blp = Blueprint( 'items', \_\_name\_\_, description="Opeations on items" )

# The Blueprint arguments are the same as the Flask Blueprint1, with an added optional description keyword argument:

# "items" is the name of the blueprint. This will be shown in the documentation

@blp.route('/item/<string:item\_id>')

class Item( MethodView ):

    def get( self, item\_id ):

        try:

            return items[item\_id], 200

        except KeyError as e:

            # return { "Error" : f"Item not Found, Exception = {e}" }, 404

            abort( 404, message= f"Item not Found, Exception = {e}" )

    def delete( self, item\_id ):

        try:

            op = items.pop( item\_id )

            return op, 202

        except KeyError as e:

            abort( 404, message= f"Item not Found having item\_id = { item\_id }" )

    @blp.arguments( ItemUpdateSchema )

    def put( self, put\_item\_data, item\_id ): # We need to keep Marshmallow argument at the 2nd place just after self.

        # put\_item\_data = request.get\_json()

        """

        if( item\_id not in items ):

            abort( 404, message= f"Item not Found having item\_id = { item\_id }" )

        if( ("item\_name" not in put\_item\_data)  and ("item\_price" not in put\_item\_data) ):

            abort( 404, message= f"API does not have required parameters" )

        """

        items[ item\_id ]["item\_name"] = put\_item\_data["item\_name"]

        items[ item\_id ]["item\_price"] = put\_item\_data["item\_price"]

        if("store\_id" in put\_item\_data):

            items[ item\_id ]["store\_id"] = put\_item\_data["store\_id"]

        return items[ item\_id ], 202

@blp.route('/addItem')

class AddItem(MethodView):

    @blp.arguments( ItemSchema )

    # def post(self):

    def post(self, item\_data): # as we are implementing Marshmallow technique. So Marshmallow will give the data to argument

                               # in order to validate data using blp.arguments( ItemSchema ) decorator and then, allows

                               # Python to proceed further if everything looks good.

        # item\_data = request.get\_json()    # As we are getting item\_data details. So, request.get\_json() is not required

        print( f"item\_data = { item\_data }")

        """

        # Not Required start : As we are implementing Marshmallow

        if( ("store\_id" not in item\_data)

            and ("item\_name" not in item\_data)

            and ("item\_price" not in item\_data) ):

            abort( 404, message= f"API does not have required parameters" )

        if( item\_data["store\_id"] not in stores ):

            # return { "Error" : f"Store not found having store\_id : { item\_data['store\_id'] }" }

            abort( 404, message= f"Store not found having store\_id : { item\_data['store\_id'] }" )

        # Not Required end : As we are implementing Marshmallow

        """

        for each\_item in items.values():

            if( ( each\_item["store\_id"] == item\_data["store\_id"] )

            and ( each\_item["item\_name"] == item\_data["item\_name"] ) ):

                abort( 404, message= f"Duplicate Item present in store\_id : { item\_data['store\_id'] }" )

        item\_id = uuid.uuid4().hex

        item\_data["item\_id"] = item\_id

        items[item\_id] = item\_data

        return item\_data, 201

@blp.route("/items")

class Items( MethodView ):

    def get( self ):

        return items, 200

############## Item.py Code End #############