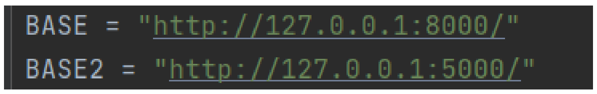
Homework1_DOS

Dependencies and App requirements:

- Requests package
- Python3
- Restful & flask are all in the requirements.txt file.
    - Pip install –r requirements.txt
    - Use pip3 for Python
- Sqlite3

Client Code:

The Client code acts as a simple terminal for user input. It takes Inputs to decide which RESTFUL link to execute first.

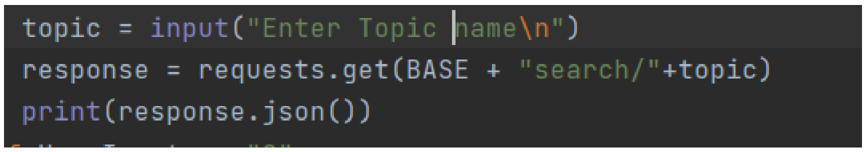


```
BASE = "http://127.0.0.1:8000/"
BASE2 = "http://127.0.0.1:5000/"
```

Base → Catalog

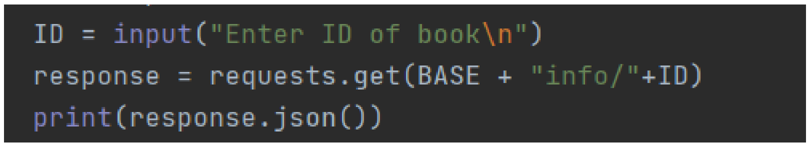
Base2→ Orderlog

- If the User enters 1 → Search by topic so the link should be the Catalog Port



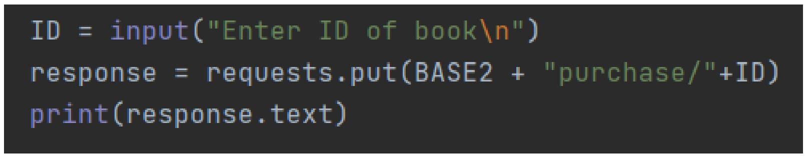
```
topic = input("Enter Topic name\n")
response = requests.get(BASE + "search/"+topic)
print(response.json())
```

- If the user enters 2 → Search by ID at Catalog port



```
ID = input("Enter ID of book\n")
response = requests.get(BASE + "info/"+ID)
print(response.json())
```

- If the user enters 3 → Purchase by ID at Orderlog port



```
ID = input("Enter ID of book\n")
response = requests.put(BASE2 + "purchase/"+ID)
print(response.text)
```

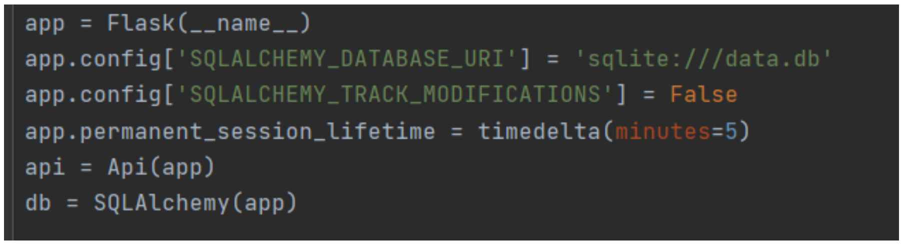
- If 4 then it exists

- We use the request import here to perform the RESTful request. The response is then changed into Json/text and printed onto the terminal

Catalog Server:

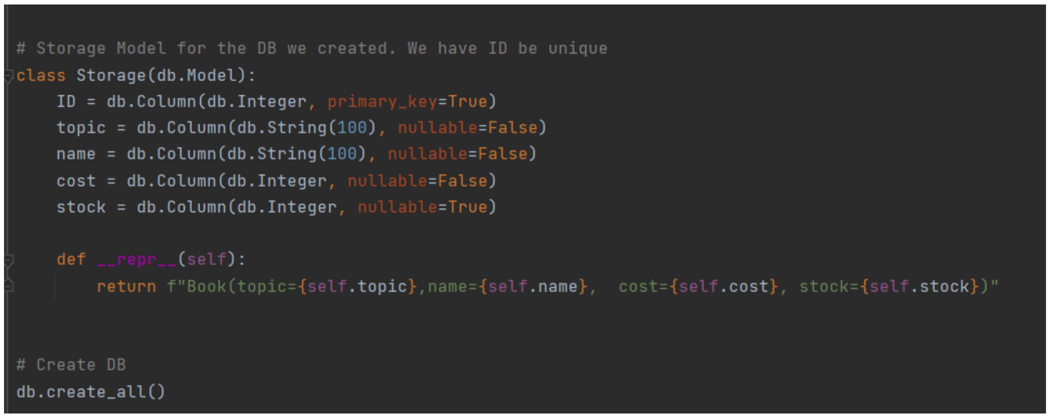
1) The Catalog server performs three operations. Being Search, Info and Update for the purchase operation from the Orderlog.

2) The Catalog Server uses an SQLITE3 DB that's connected to SQLAlchemy, make sure the **the data.db is placed within the same folder as the Python file**



```python
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.permanent_session_lifetime = timedelta(minutes=5)
api = Api(app)
db = SQLAlchemy(app)
```

3) We define the Model for the Database Table as follows:

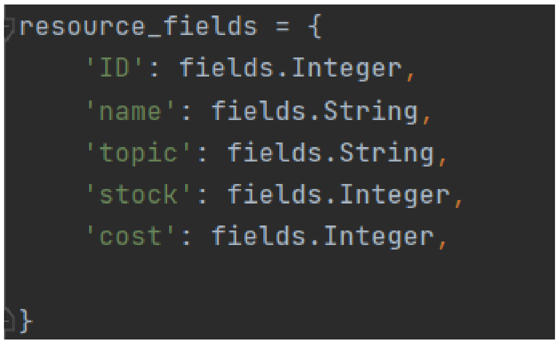


```python
# Storage Model for the DB we created. We have ID be unique
class Storage(db.Model):
    ID = db.Column(db.Integer, primary_key=True)
    topic = db.Column(db.String(100), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    cost = db.Column(db.Integer, nullable=False)
    stock = db.Column(db.Integer, nullable=True)

    def __repr__(self):
        return f"Book(topic={self.topic},name={self.name},  cost={self.cost}, stock={self.stock})"


# Create DB
db.create_all()
```

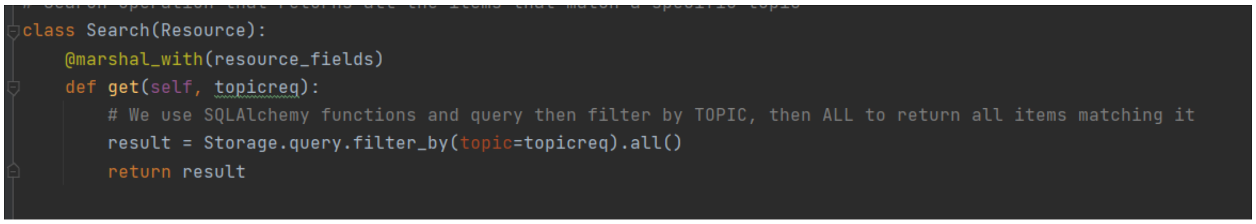
The Database format isn't serializable into Json in it's current state so we use marshall with library to help us perform that.



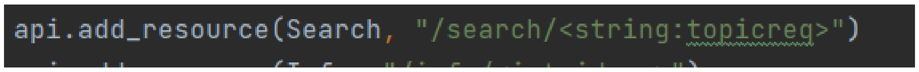
```python
resource_fields = {
    'ID': fields.Integer,
    'name': fields.String,
    'topic': fields.String,
    'stock': fields.Integer,
    'cost': fields.Integer,

}
```

1- Search Operation: We perform Query on the DB then return all the lines that match the attribute. We take an argument that’s Topicreq and use it as the parameter for the query since that’s the topic we wanna match

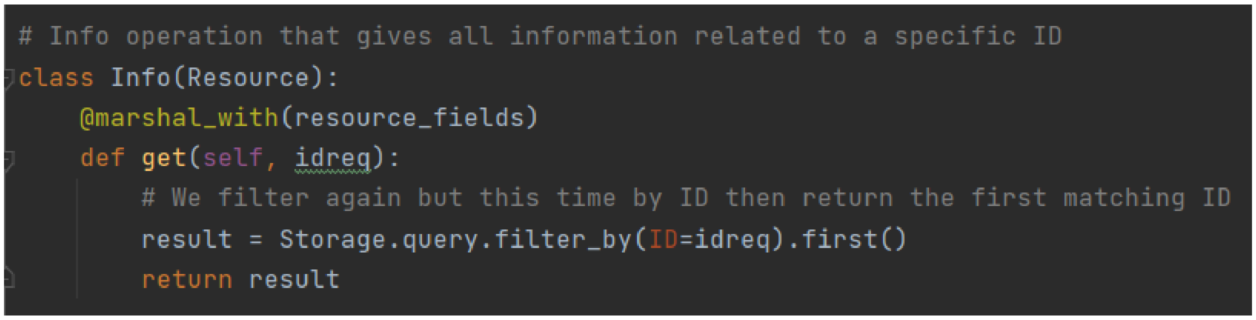


```python
class Search(Resource):
    @marshal_with(resource_fields)
    def get(self, topicreq):
        # We use SQLAlchemy functions and query then filter by TOPIC, then ALL to return all items matching it
        result = Storage.query.filter_by(topic=topicreq).all()
        return result
```

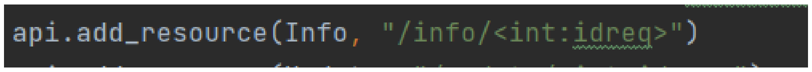


```python
api.add_resource(Search, "/search/<string:topicreq>")
```

2- Info Operation: Performs the same thing as Search except this time we are looping and stopping on the very first result. We send the ID that’s requested from the client and query based on it.

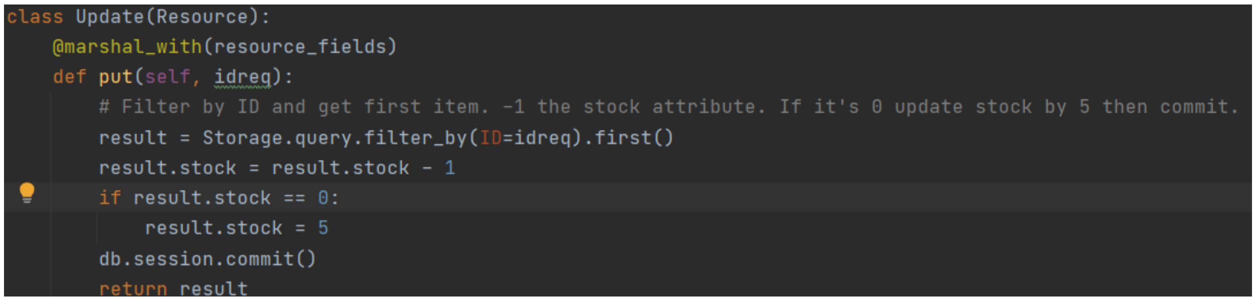


```python
# Info operation that gives all information related to a specific ID
class Info(Resource):
    @marshal_with(resource_fields)
    def get(self, idreq):
        # We filter again but this time by ID then return the first matching ID
        result = Storage.query.filter_by(ID=idreq).first()
        return result
```

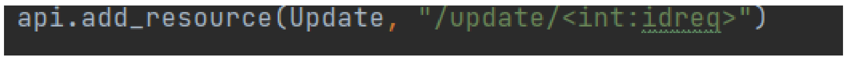


```python
api.add_resource(Info, "/info/<int:idreq>")
```

3- Update Operation: This Operation is called by orderlog. It queries based on ID then takes the Stock attribute and -1s it. If the stock is empty we replenish it by 5



```python
class Update(Resource):
    @marshal_with(resource_fields)
    def put(self, idreq):
        # Filter by ID and get first item. -1 the stock attribute. If it's 0 update stock by 5 then commit.
        result = Storage.query.filter_by(ID=idreq).first()
        result.stock = result.stock - 1
        if result.stock == 0:
            result.stock = 5
        db.session.commit()
        return result
```

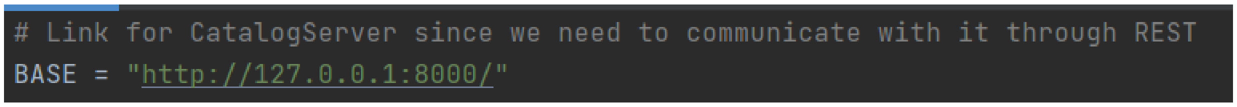


```python
api.add_resource(Update, "/update/<int:idreq>")
```

The element in the table is returned to Orderlog then after committing the changes in Catalog DB

Orderlog.py:

We have the link to the Catalog in order to call the Update function from Catalog Server.

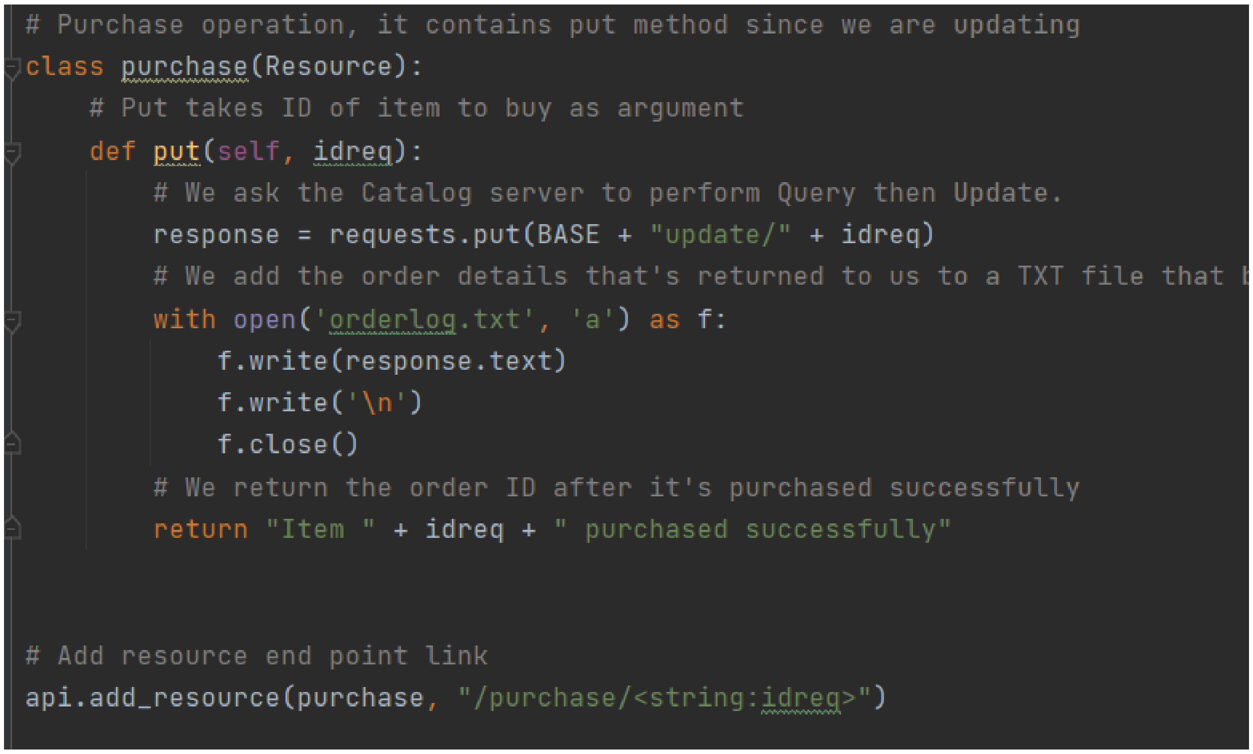


```
# Link for CatalogServer since we need to communicate with it through REST
BASE = "http://127.0.0.1:8000/"
```

Purchase Operation:

We use put instead of GET since we are updating. Then in order to store the element we received from the Catalog server we create/open a TXT file where we append the Data into it in the form of an orderlog.

The Client receives the Item name+ it being purchased successfully



```
# Purchase operation, it contains put method since we are updating
class purchase(Resource):
    # Put takes ID of item to buy as argument
    def put(self, idreq):
        # We ask the Catalog server to perform Query then Update.
        response = requests.put(BASE + "update/" + idreq)
        # We add the order details that's returned to us to a TXT file that 
        with open('orderlog.txt', 'a') as f:
            f.write(response.text)
            f.write('\n')
            f.close()
        # We return the order ID after it's purchased successfully
        return "Item " + idreq + " purchased successfully"


# Add resource end point link
api.add_resource(purchase, "/purchase/<string:idreq>")
```

In order to run the Code. Run the Servers first each in a separate Terminal/Machine then run the Client on a separate machine then interact with the Client and watch the requests reach the servers on the other terminals.

The Output sample shows it best