Python Developer Task
Product Inventory REST API – v0.1

Summary

RESTful API for remote client consumption, using Python Flask framework

Links

- https://github.com/ankesand/product_api_demo
- http://www.ankesand.com/work/product-demo/api

Features

- Live demo environment
- Database backend – MySQL / SQL-Alchemy

Deficiencies

- Unauthenticated
- ‘Dockerisation’

Design

Requirements

Capability	HTTP Method	Argument(s)	Argument Type	Endpoint
Register a product	POST	SKU Name Qty Price	Query Query Query Query	/api/products
Retrieve Product Details from SKU	GET	sku	Path	/api/products/[SKU]
List all available products (>0 Qty)	GET	available	Path	/api/products/available
List all sold out products (0 Qty)	GET	sold-out	Path	/api/products/sold-out
Register Qty Change (SKU, +/- Value)	PUT	SKU plus OR minus Qty	Path Query Query	/api/products/[SKU]

Overview

- MySQL database (“product_demo”)
  - Table (“products”)
    - Fields (“SKU”, “Name”, “Qty”, “Price”)
- SQLAlchemy model (“Products”)
  - Class (“Product”)
    - Attributes (“sku”, “name”, “qty”, “price”)
- Flask API
  - Routes / Views (“/api”)
    - Methods (“GET”, “POST”, “PUT”)
    - URI parameters
      - Path (“sku”)
      - Query (GET: “available”, “sold-out”; POST: “SKU”, “Name”, “Qty”, “Price”; PUT: “SKU”, “plus”, “minus”)

Build

MySQL

```
mysql> create database product_demo;
Query OK, 1 row affected (0.29 sec)

mysql> use product_demo;
Database changed
mysql> create table products
    -> (
    -> SKU VARCHAR(255) PRIMARY KEY,
    -> Name VARCHAR(255) NOT NULL,
    -> Qty INT(32) NOT NULL,
    -> Price FLOAT NOT NULL
    -> );
Query OK, 0 rows affected, 1 warning (0.25 sec)

mysql> describe products;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| SKU   | varchar(255) | NO   | PRI | NULL    |       |
| Name  | varchar(255) | NO   |     | NULL    |       |
| Qty   | int(32)      | NO   |     | NULL    |       |
| Price | float        | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

mysql> insert into products values
    -> ('2345', 'Apple iPhone 5', 5, 59.99),
    -> ('6543', 'Apple iPhone 6', 0, 64.0),
    -> ('9845', 'Apple iPhone 6s', 1, 69.99),
    -> ('2347', 'Apple iPhone 7', 10, 74.99)
    -> ;
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from products;
+------+-----------------+-----+-------+
| SKU  | Name            | Qty | Price |
+------+-----------------+-----+-------+
| 2345 | Apple iPhone 5  |   5 | 59.99 |
| 2347 | Apple iPhone 7  |  10 | 74.99 |
| 6543 | Apple iPhone 6  |   0 |    64 |
| 9845 | Apple iPhone 6s |   1 | 69.99 |
+------+-----------------+-----+-------+
4 rows in set (0.00 sec)

mysql>
```

Python (SQLAlchemy)

```
from sqlalchemy import MetaData, Table, Column, String, Integer, Float, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from config import MYSQL_SERVER

mysql_server = MYSQL_SERVER

metadata = MetaData()

Products = Table("products",
                 metadata,
                 Column("SKU", String(255), primary_key=True),
                 Column("Name", String(255)),
                 Column("Qty", Integer()),
                 Column("Price", Float())
                 )

Base = declarative_base()

class Product(Base):

    __tablename__ = "products"

    sku = Column("SKU", String(255), primary_key=True)
    name = Column("Name", String(255))
    qty = Column("Qty", Integer())
    price = Column("Price", Float())

    def __init__(self, sku, name, qty, price):

        self.sku = sku
        self.name = name
        self.qty = qty
        self.price = price

engine = create_engine(mysql_server)

metadata.create_all(engine)

demo_data = [
              {
                "SKU": "2345",
                "Name": "Apple iPhone 5",
                "Qty": 5,
                "Price": 59.99,
               },
              {
                "SKU": "6543",
                "Name": "Apple iPhone 6",
                "Qty": 0,
                "Price": 64.00,
               },
              {
              "SKU": "9845",
              "Name": "Apple iPhone 6s",
              "Qty": 1,
              "Price": 69.99,
               },
              {
              "SKU": "2347",
              "Name": "Apple iPhone 7",
              "Qty": 10,
              "Price": 74.99,
               },
             ]
```

```
session = sessionmaker(bind=engine)()

for product in demo_data:
    try:
        session.add(Product(product["SKU"],
                            product["Name"],
                            product["Qty"],
                            product["Price"]
                            )
                    )
        session.commit()
        print("Added: " + product["Name"])
    except:
        session.rollback()
        print("Could not add: " + product["Name"])

[row.sku for row in session.query(Product.sku).all()]
```

Usage

Register a product

```
import requests

# Register a product

>>> r = requests.post("http://www.ankesand.com/work/product-demo/api?SKU=3210&Name=Nokia
3210&Qty=1&Price=32.1")
>>> r.text
"Success: Added {'SKU': '3210', 'Name': 'Nokia 3210', 'Qty': '1', 'Price': '32.1'}"

# Check product registered

>>> r = requests.get("http://www.ankesand.com/work/product-demo/api/3210")
>>> r.text
"{'SKU': '3210', 'Name': 'Nokia 3210', 'Qty': 1, 'Price': 32.1}"
>>>
```

Retrieve Product Details from SKU

```
# Retrieve Product Details from SKU

>>> r = requests.get("http://www.ankesand.com/work/product-demo/api/2345")
>>> r.text
"{'SKU': '2345', 'Name': 'Apple iPhone 5', 'Qty': 5, 'Price': 59.99}"
>>>
```

List all available products (>0 Qty)

```
# List all available products (>0 Qty)

>>> r = requests.get("http://www.ankesand.com/work/product-demo/api/available")
>>> r.text
"[{'SKU': '2345', 'Name': 'Apple iPhone 5', 'Qty': 5, 'Price': 59.99}, {'SKU': '2347',
'Name': 'Apple iPhone 7', 'Qty': 10, 'Price': 74.99}, {'SKU': '3210', 'Name': 'Nokia
3210', 'Qty': 1, 'Price': 32.1}, {'SKU': '9845', 'Name': 'Apple iPhone 6s', 'Qty': 1,
'Price': 69.99}]"
>>>
```

List all sold out products (0 Qty)

```
# List all sold out products (0 Qty)

>>> r = requests.get("http://www.ankesand.com/work/product-demo/api/sold-out")
>>> r.text
"[{'SKU': '6543', 'Name': 'Apple iPhone 6', 'Qty': 0, 'Price': 64.0}]"
>>>
```

Register Qty Change (SKU, +/- Value)

```
# Register Qty Change (SKU, +/- Value)

>>> r = requests.put("http://www.ankesand.com/work/product-demo/api/3210?plus=1")
>>> r.text
'Updated quantity from: 1 to: 2'
>>>

# Check quantity change registered

>>> r = requests.get("http://www.ankesand.com/work/product-demo/api/3210")
>>> r.text
"{'SKU': '3210', 'Name': 'Nokia 3210', 'Qty': 2, 'Price': 32.1}"
>>>
```

Notes for discussion

- Flask-Django migration
- In-memory data
- Authentication (password / token)
- Virtual environment(s)
- Git push/pull
- Double Vs Float [negative values?!]
- NOT NULL
- Requests – URI parameters vs data