Statistical foundation of Data Sciences

Practical- 11

Roll number: GF202344767

Name: Ishita Mehta

Workflow and Interpretations

1. Defining the Data Source (Inventory Dataset)

The first step is deciding **where your data will come from**.
For a beginner-level API, the simplest option is to use a **Python list of dictionaries** that represents the inventory.

Why this method?

- It removes the need for a database initially.
- Very easy to edit, update, and test.
- Perfect for learning how APIs work before connecting MySQL.

What it stores?

Each dictionary holds:

- Product ID
- Product Name
- Quantity
- Price

These fields reflect common attributes needed in any inventory management system.

2. Choosing Flask as the Backend Framework

Flask is selected because:

- It is lightweight and minimal (good for a small API).
- Easy to run inside **Jupyter Notebook**.
- Requires very little configuration.
- Perfect for beginners learning backend development.

Flask allows mapping URLs (routes) to functions that return data.
This makes it ideal for building REST-style APIs.

3. Designing REST Endpoints

Two endpoints are required based on your project requirement:

A. Endpoint 1: List All Inventory Items

Purpose:

To give the user or frontend a complete view of all products.

Why this endpoint?

- It acts like the “homepage” of your inventory system.
- Useful for dashboards that display the entire stock list.

Output behaviour:

Returns all items in the inventory dataset in a structured JSON format.

B. Endpoint 2: Retrieve a Single Product by ID

Purpose:

To get detailed information about one specific product.

Why use the product ID?

- IDs ensure unique identification.
- Faster and more reliable than searching by name.
- Matches how real inventory systems work (SKU numbers, product IDs).

Logic behind this endpoint:

- The API checks if the ID exists in the inventory list.
- If found → Product details are returned.
- If not found → API returns a clear “not found” error message.

This helps avoid confusion and improves the robustness of the system.

4. Running Flask in Jupyter Notebook

Since you're using Jupyter Notebook, you cannot run Flask normally (it blocks the notebook). Therefore, the API is run **in a separate background thread**.

Why use a background thread?

- It prevents the notebook from freezing.
- You can continue running other cells while the server is active.
- Easy to stop and restart by interrupting the kernel.

This makes development smoother.

5. Testing the API Endpoints

Once the server is running, each endpoint is tested using a browser or tools like Postman.

Why testing matters?

- Ensures routes are working properly.
- Checks that JSON responses are correct.
- Helps verify that the correct HTTP status codes are returned (e.g., 200 OK, 404 Not Found).

Testing validates the entire API logic.

6. Potential Extensions (Future Workflow Steps)

Once the base API works, the project can evolve:

A. Connect to MySQL

To store inventory data persistence rather than in memory.

B. Add CRUD Operations

- Create new product
- Update existing product
- Delete a product
  This makes the API fully functional.

C. Add Authentication

Only authorized users can modify inventory data.

D. Add Frontend (HTML/JS or React)

To visually display inventory data.

7. Stopping the Server

Since Jupyter Notebook is used, the server is stopped by **interrupting the kernel**.
This resets the notebook state and stops the Flask process.

GitHub Repository link

https://github.com/pineapplesdontbelongonpizza/CSU1658_practical1_Testing_Pandas_and_Numpy.git