1. **Fast API Python code:**

from fastapi import FastAPI, HTTPException

from pymongo import MongoClient

from pydantic import BaseModel

import requests

import os

from dotenv import load\_dotenv

from bson import ObjectId  # Import for handling ObjectId in MongoDB

load\_dotenv()

app = FastAPI()

# MongoDB Connection

client = MongoClient("mongodb://localhost:27017/")

db = client.inventory\_db

collection = db.products

class Product(BaseModel):

    Product\_ID: str  # Match MongoDB "Product ID"

    Name: str

    Unit\_Price: float  # Match MongoDB "Unit Price"

    Stock\_Quantity: int  # Match MongoDB "Stock Quantity"

    Description: str

# 1. Get single product by ID

@app.get("/getSingleProduct/{product\_id}")

def get\_single\_product(product\_id: str):  # Keep product\_id as string

    product = collection.find\_one({"Product ID": product\_id}, {"\_id": 0})  # Search in "Product ID" field

    if not product:

        raise HTTPException(status\_code=404, detail="Product not found")

    return product

# 2. Get all products

@app.get("/getAll")

def get\_all():

    return list(collection.find({}, {"\_id": 0}))

# 3. Add a new product

@app.post("/addNew")

def add\_product(product: Product):

    # Check if the product already exists

    if collection.find\_one({"Product\_ID": product.Product\_ID}):

        raise HTTPException(status\_code=400, detail="Product with this ID already exists")

    # Insert product into MongoDB

    collection.insert\_one(product.dict())

    return {"message": "Product added successfully", "product": product}

# 4. Delete a product by ID

@app.get("/deleteOne/{product\_id}")

def delete\_product(product\_id: str):  # Ensure product\_id is a string

    result = collection.delete\_one({"Product ID": product\_id})  # Match the MongoDB field name

    if result.deleted\_count == 0:

        raise HTTPException(status\_code=404, detail="Product not found")

    return {"message": f"Product {product\_id} deleted successfully"}

# 5. Find products that start with a letter

@app.get("/startsWith/{letter}")

def starts\_with(letter: str):

    return list(collection.find({"Name": {"$regex": f"^{letter}", "$options": "i"}}, {"\_id": 0}))

# 6. Paginate products

@app.get("/paginate/{start\_id}/{end\_id}")

def paginate(start\_id: str, end\_id: str):

    products = list(collection.find(

        {"Product ID": {"$gte": start\_id, "$lte": end\_id}},  # Match "Product ID" as a string

        {"\_id": 0}  # Exclude MongoDB ObjectId

    ).sort("Product ID", 1).limit(10))  # Sort in ascending order

    if not products:

        raise HTTPException(status\_code=404, detail="No products found in this range")

    return products

# 7. Convert product price from USD to EUR

@app.get("/convert/{product\_id}")

def convert\_price(product\_id: str):  # Change product\_id to string

    product = collection.find\_one({"Product ID": product\_id})  # Match "Product ID" field

    if not product:

        raise HTTPException(status\_code=404, detail="Product not found")

    response = requests.get("https://api.exchangerate-api.com/v4/latest/USD")

    exchange\_rate = response.json()["rates"]["EUR"]

    converted\_price = product["Unit Price"] \* exchange\_rate  # Fix key to "Unit Price"

    return {

        "Product": product["Name"],

        "Price in EUR": round(converted\_price, 2)

    }

1. **Unit test code:**

from fastapi.testclient import TestClient

from main import app  # Ensure this imports your FastAPI app correctly

client = TestClient(app)

# 1. Test fetching all products

def test\_get\_all():

    response = client.get("/getAll")

    assert response.status\_code == 200

    assert isinstance(response.json(), list)

# 2. Test adding a new product

def test\_add\_new\_product():

    new\_product = {

        "Product\_ID": "AUTO999",

        "Name": "Test Product",

        "Unit\_Price": 199.99,

        "Stock\_Quantity": 10,

        "Description": "A sample test product"

    }

    response = client.post("/addNew", json=new\_product)

    assert response.status\_code == 200

    assert response.json()["message"] == "Product added successfully"

# 3. Test fetching a single product

def test\_get\_single\_product():

    response = client.get("/getSingleProduct/AUTO999")

    assert response.status\_code == 200

    assert response.json()["Product\_ID"] == "AUTO999"

# 4. Test deleting a product

def test\_delete\_product():

    response = client.delete("/deleteOne/AUTO999")

    assert response.status\_code == 200

    assert response.json()["message"] == "Product AUTO999 deleted successfully"

# 5. Test pagination endpoint with valid IDs

def test\_paginate():

    response = client.get("/paginate/2/10")  # Ensure products exist in this range

    assert response.status\_code == 200

    assert isinstance(response.json(), list)

# 6. Test convert price endpoint with valid product ID

def test\_convert\_price():

    response = client.get("/convert/AUTO999")

    assert response.status\_code in [200, 404]  # It could return 404 if product doesn't exist

1. **Screenshots of all the URL endpoints for your API being run.**