Inventory Management Tool (Backend APIs)

Project Overview

Build a small backend application to manage inventory for a small business. The app should expose REST APIs to manage users and products. The primary features should include:

- User authentication (Login)
- Product addition
- Product quantity updates
- Get products and their inventories

Expected Deliverables

- A working backend server with REST APIs for the above functionality
- Database schema
- OpenAPI/Swagger documentation for all endpoints
- Setup documentation (README)
- Sample Postman collection for testing
- Test your code using the script below

Feature Requirements

1. User Authentication

- **Endpoint:** `POST /login`
- **Request:**
-

  JSON

  ```
  {
    "username": "string",
    "password": "string"
  }
  ```

  Response:
  - On success: JWT token or session cookie
  - On failure: Error message

- **Authentication:** Use JWT (preferred) or session-based login

2. Add Product

- **Endpoint:** `POST /products`
- **Payload:**
- Unset

```
{
  "name": "string",
  "type": "string",
  "sku": "string",
  "image_url": "string",
  "description": "string",
  "quantity": integer,
  "price": number
}
```

  Authentication required: Yes
- **Response:** Product ID and confirmation

3. Update Product Quantity

- **Endpoint:** `PUT /products/{id}/quantity`
- **Payload:**
- Unset `{"quantity": integer}` **Response:** Updated product details or confirmation message
- **Authentication required:** Yes

4. Get Products

- **Endpoint:** `GET /products`
- **Request parameters: Paginate the API call.**
- **Response:** List of all products
- **Authentication required:** Yes

Stretch Goals (Optional)

- Admin portal
- Basic analytics (e.g., most added products)
- Basic frontend with React or Vue
- Dockerize the application

Project Tasks Breakdown

1. Design database schema
2. Set up backend project
3. Implement product addition API
4. Implement update quantity API
5. Implement user auth and login API
6. Test server API, error handling, edge cases
7. Write API docs, README
8. (Optional) Do stretch goals

Submission Guidelines

The candidate should submit:

- GitHub link of the codebase
- The GitHub repo must have setup instructions in `README.md`
- API documentation (Markdown or Swagger/OpenAPI)
- Database initialization script to create schema in the database.

Notes for Candidates

- Use any language and framework you're comfortable with.
- Use any database you're comfortable with such as PostgresQL, SQLite etc.
- Keep the project simple, functional, and secure.
- You are free to use third-party libraries for things like JWT or password hashing.
- If stuck or short on time, document assumptions and partial implementation.
- Clean code matters more than feature completeness.

Please refer to the following script to test your code. Instructions are provided at the beginning of the file.

```
"""
Inventory Management Tool - API Test Script
This Python script is provided to test your Inventory Management Tool as part of the Fi Internship Assignment.
Requirements:
Python 3.6+
`requests` library
Setup Instructions:
1. Install Python dependencies:
Make sure you have requests installed. If not, run:
`pip install requests`
2. Set your server URL:
Open test_api.py in a text editor and update the BASE_URL variable to point to your running server instance:
3. Run the script:
From your terminal, run:
python test_api.py
"""

import requests

BASE_URL = "http://localhost:8080"  # Change this to your API base URL

def print_result(test_name, passed, expected=None, got=None, request_data=None, response_body=None):
    """
    Prints test result.
    If passed, prints only success.
    If failed, prints request, expected vs got, and response body.
    """
    if passed:
```

```
        print(f"{test_name}: PASSED")
    else:
        print(f"{test_name}: FAILED")
        if request_data:
            print(f"  Request: {request_data}")
        if expected is not None and got is not None:
            print(f"  Expected: {expected}, Got: {got}")
        if response_body:
            print(f"  Response Body: {response_body}")

def test_register_user():
    """
    Change payload keys/values as needed for your registration API.
    Expected status codes are 201 (created) or 409 (conflict if user exists).
    """
    payload = {"username": "puja", "password": "mypassword"}  # Change username/password if needed
    res = requests.post(f"{BASE_URL}/register", json=payload)
    passed = res.status_code in [201, 409]
    print_result("User Registration", passed, "201 or 409", res.status_code, payload, res.text)

def test_login():
    """
    Change payload for different username/password.
    On success, expects 200 status and an 'access_token' in JSON response.
    Returns the token for authenticated requests.
    """
    payload = {"username": "puja", "password": "mypassword"}  # Change to test different login
credentials
    res = requests.post(f"{BASE_URL}/login", json=payload)
    token = None
    passed = False
    if res.status_code == 200:
        try:
            token = res.json().get("access_token")
            passed = token is not None
        except Exception:
            passed = False
    print_result("Login Test", passed, {"username": payload["username"], "password":
payload["password"]}, res.text, payload, res.text)
    return token

def test_add_product(token):
    """
    Change payload fields as per your product API requirements.
    Must include Authorization header with Bearer token.
    Returns product_id on success to be used in other tests.
    """
    payload = {
        "name": "Phone",          # Change product name
        "type": "Electronics",    # Change type/category
        "sku": "PHN-001",         # Change SKU if needed
        "image_url": "https://example.com/phone.jpg",  # Change image URL
```

```
        "description": "Latest Phone",                # Change description
        "quantity": 5,            # Initial quantity
        "price": 999.99           # Price
    }
    res = requests.post(f"{BASE_URL}/products", json=payload, headers={"Authorization": f"Bearer
{token}"})
    passed = res.status_code == 201
    if passed:
        print("Add Product: PASSED")
        try:
            return res.json().get("product_id")
        except Exception:
            return None
    else:
        print_result("Add Product", False, 201, res.status_code, payload, res.text)
        return None

def test_update_quantity(token, product_id, new_quantity):
    """
    Tests update quantity API for a specific product.
    Change endpoint if your API uses a different URL structure.
    Pass the product ID and the new quantity.
    """
    payload = {"quantity": new_quantity}  # Change field name if your API expects different key
    res = requests.put(
        f"{BASE_URL}/products/{product_id}/quantity",
        json=payload,
        headers={"Authorization": f"Bearer {token}"}
    )
    passed = res.status_code == 200
    if passed:
        if res.text:
            try:
                updated_info = res.json()
                updated_qty = updated_info.get("quantity", "unknown")  # Change key if API uses a different
key for quantity
                print(f"Update Quantity: PASSED, Updated quantity: {updated_qty}")
            except Exception:
                print("Update Quantity: PASSED, but response body is not valid JSON")
        else:
            print("Update Quantity: PASSED, but response body is empty")
    else:
        print_result("Update Quantity", False, 200, res.status_code, payload, res.text)

def test_get_products(token, expected_quantity):
    """
    Tests fetching the list of products.
    Change endpoint if needed.
    Checks if there is a product named 'Phone' with expected quantity.
    Change 'name' and 'quantity' keys if your API structure differs.
    """
    res = requests.get(f"{BASE_URL}/products", headers={"Authorization": f"Bearer {token}"})
```

```
    if res.status_code != 200:
        print_result("Get Products", False, 200, res.status_code, None, res.text)
        return

    try:
        products = res.json()
    except Exception:
        print_result("Get Products", False, "valid JSON list", "Invalid JSON", None, res.text)
        return

    phone_products = [p for p in products if p.get("name") == "Phone"]
    if not phone_products:
        print("Get Products: FAILED")
        print("  Could not find product named 'Phone'")
        print(f"  Response Body: {products}")
        return

    phone_quantity = phone_products[0].get("quantity")
    if phone_quantity == expected_quantity:
        print(f"Get Products: PASSED (Quantity = {phone_quantity})")
    else:
        print("Get Products: FAILED")
        print(f"  Expected Quantity: {expected_quantity}, Got: {phone_quantity}")
        print(f"  Response Body: {products}")

def run_all_tests():
    """
    Runs all tests in sequence.
    If any test fails, subsequent tests are skipped.
    """
    test_register_user()
    token = test_login()
    if not token:
        print("Login failed. Skipping further tests.")
        return

    product_id = test_add_product(token)
    if not product_id:
        print("Product creation failed. Skipping further tests.")
        return

    new_quantity = 15  # Change this to test different updated quantity
    test_update_quantity(token, product_id, new_quantity)
    test_get_products(token, expected_quantity=new_quantity)

if __name__ == "__main__":
    run_all_tests()
```