

Introduction:

My name is Muhammad Umer, currently a BSCS student at FAST-NUCES, Karachi. I'm passionate about technology, especially in the areas of software development. I have hands-on experience with C++,python and web development, and I've completed internships at Developershub Corporation and Saiket System, where I worked on real-world projects that strengthened my technical and problem-solving skills.

What excites me about Bazaar is how it's digitizing traditional commerce in Pakistan. I'd love to be part of a fast-paced team where I can contribute my technical skills, grow professionally, and work on meaningful projects that impact millions of users.

Stage 1: Inventory Tracking System – Single Kiryana Store (Backend API)

Goal:

Build a simple, backend inventory tracking system for a single kiryana store, with future scalability in mind for multi-store, multi-supplier support.

Approach:

I developed a Flask + SQLite backend to manage inventory for a single kiryana store. I created two models: Product (with unique SKU) and StockMovement (to log stock-in, sales, and removals). RESTful APIs were built to add/view products and stock movements. Data is stored in inventory.db using SQLAlchemy, and the system was tested via Postman and CLI for both success and error handling.

Data Models:

1.Product Model

Stores product details like name, sku, category, and price.

Enforces unique SKU constraint to avoid duplicates.

2.StockMovement Model

Tracks all inventory actions:

stock-in, sale, manual-removal.

Each entry is linked to a product via product_id.

```
PS C:\Users\Umer\OneDrive\Desktop> python app.py
```

```
* Serving Flask app 'app'
```

```
* Debug mode: on
```

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
```

```
* Running on http://127.0.0.1:5000
```

```
Press CTRL+C to quit
```

```
* Restarting with stat
```

```
* Debugger is active!
```

```
* Debugger PIN: 138-737-945
```

```
127.0.0.1 - - [11/Apr/2025 20:18:52] "GET /stock-movement HTTP/1.1" 200 -
```

```
127.0.0.1 - - [11/Apr/2025 20:21:00] "POST /products HTTP/1.1" 400 -
```

```
127.0.0.1 - - [11/Apr/2025 20:21:08] "GET /products HTTP/1.1" 200 -
```

New Collection / http://127.0.0.1:5000/stock-movement

GET http://127.0.0.1:5000/stock-movement

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key
Key

New Collection / http://127.0.0.1:5000/products

GET http://127.0.0.1:5000/products

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key
Key

New Collection / http://127.0.0.1:5000/stock-movement

POST http://127.0.0.1:5000/stock-movement

Params Authorization Headers (9) Body Scripts Settings

Query Params

Key
Key

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
1 [
2   {
3     "id": 1,
4     "movement_type": "stock-in",
5     "product_id": 1,
6     "quantity": 10
7   }
8 ]
```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
1 [
2   {
3     "category": "Grains",
4     "id": 1,
5     "name": "Rice",
6     "price": 200.0,
7     "sku": "RICE123"
8   },
9   {
10    "category": "Electronics",
11    "id": 2,
12    "name": "Laptop",
13    "price": 1200.99,
14    "sku": "LAP123"
15  }
16 ]
```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
1 {
2   "message": "Stock movement recorded successfully"
3 }
```

بازار
BAZAAR

```
* Restarting with stat
```

```
* Debugger is active!
```

```
* Debugger PIN: 138-737-945
```

```
127.0.0.1 - - [11/Apr/2025 20:18:52] "GET /stock-movement HTTP/1.1" 200 -
```

```
127.0.0.1 - - [11/Apr/2025 20:21:00] "POST /products HTTP/1.1" 400 -
```

```
127.0.0.1 - - [11/Apr/2025 20:21:08] "GET /products HTTP/1.1" 200 -
```

```
127.0.0.1 - - [11/Apr/2025 20:27:55] "GET /products HTTP/1.1" 200 -
```

```
127.0.0.1 - - [11/Apr/2025 20:28:07] "GET /stock-movement HTTP/1.1" 200 -
```

```
127.0.0.1 - - [11/Apr/2025 20:28:18] "GET /stock-movement HTTP/1.1" 200 -
```

```
C:\Users\Umer\OneDrive\Desktop\app.py:67: LegacyAPIWarning: The Query.get() method is deprecated since: 2.0 (Background on SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
```

```
product = Product.query.get(data['product_id'])
```

```
127.0.0.1 - - [11/Apr/2025 20:28:29] "POST /stock-movement HTTP/1.1" 201 -
```

API Endpoints Implemented:


Endpoint	Method	Purpose
/products	POST	Add a new product (with SKU uniqueness check)
/products	GET	Fetch all products
/stock-movement	POST	Record a stock movement for an existing product
/stock-movement	GET	View all stock movement records



Code Quality Features:

Checks for existing SKU to maintain data integrity.Returns appropriate HTTP status codes (201, 400, 404).Uses @app.route() decorators for clean, RESTful structure.

Terminal Summary:

 Terminal output with Flask API running locally at http://127.0.0.1:5000
Shows API logs: GET and POST requests to /products and /stock-movement

Postman tests:

GET/products shows product list (e.g., Rice, Laptop)
POST/stock-movement successfully records a stock-in movement
Response confirms: "message": "**Stock movement recorded successfully.**"

Stage2:

```
C:\Users\User\OneDrive\Desktop>stage2.py python stage2.py
C:\Users\User\AppData\Roaming\Python\Python312\site-packages\flask_limiter\extension.py:316:
RuntimeWarning: Running on http://127.0.0.1:5000
Press CTRL-C to quit
* Restarting with stat
C:\Users\User\AppData\Roaming\Python\Python312\site-packages\flask_limiter\extension.py:316:
RuntimeWarning: This is not recommended for production use. See: https://flask-limiter.readthedocs.io/en/latest/warnings.warn#
* Debugger is active!
* Debugger PIN: 138-737-945
```

1. Register API:

Method:POST

URL:http://127.0.0.1:5000/register

```
json
{
  "username": "umer",
  "password": "password123"
}
json
{
  "message": "User registered successfully!"
}
```

2. Login API:

Method:POST

URL:http://127.0.0.1:5000/login

```
{  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ..."  
}
```

3. Stock Movement API:

Method: POST

URL: http://127.0.0.1:5000/stock-movement

Headers:

Key	Value
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
Content-Type	application/json
json	
{	
"product_id": 1,	{
"movement_type": "in",	"message": "stock movement recorded successfully!"
"quantity": 20	}
}	

Objective of Stage 2:

Implement user authentication, secure access, and basic CRUD APIs for products, stores, and stock. Enable JWT-based login, rate limiting, and stock tracking per store.

Tech Stack Used:

- FastAPI + SQLAlchemy (Product/Store/Stock APIs)
- Flask + SQLAlchemy (Authentication, Rate Limiting)
- PostgreSQL as Database

JWT for Secure User Sessions

Flask-Limiter for Rate Limiting

Key Functionalities & Approach:

API Functionalities:

Product/Store/Stock APIs (via FastAPI)

Stock per Store filter

User Registration & Login (via Flask)

Token-Protected Stock Movement

Password Hashing (Secure Storage)

Approach:

Used Pydantic models for input validation (FastAPI)

Used Flask decorators for JWT token verification

Integrated both Flask (auth) and FastAPI (inventory logic) in one backend structure

Applied REST principles for clear route handling

Stage 3:

```
PS C:\Users\Umer\OneDrive\Desktop\Stage3> uvicorn main:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\Umer\\OneDrive\\Desktop\\Stage3']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [10836] using StatReload
C:\Users\Umer\OneDrive\Desktop\Stage3\main.py:23: RuntimeWarning: coroutine 'FastAPILimiter.init' was never awaited
FastAPILimiter.init(app)
RuntimeWarning: Enable tracemalloc to get the object allocation traceback
INFO: Started server process [23388]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Goal:

- Thousands of stores
- Concurrent operations
- Near real-time stock sync
- Audit logs
- Horizontal scalability
- Async/event-driven updates
- Read/write separation
- Caching & rate-limits

Scalable Architecture:

FastAPI (async): High concurrency via non-blocking routes.

RabbitMQ (aio_pika): Queues stock updates, enabling async, decoupled processing.

Celery (optional): Parallel processing with scalable workers.

Async & Event-Driven:

POST /update-stock/: Publishes to RabbitMQ.

Uses asyncpg for fast DB writes.

Celery can handle retryable background updates.

Read/Write DB Split:

get_session(is_read=True/False) connects to replica/master.

Prevents overload, enables scalable reads.

Redis Caching:

GET /get-stock/{id} checks Redis first → fast, low-latency.

Falls back to DB on cache miss.

Rate Limiting:

fastapi-limiter restricts API usage (e.g. 5 reqs/min/client).

Observability:

/metrics via Prometheus tracks latency, errors.

Grafana dashboard-ready.

Audit Logging:

Log: Product {id} updated to {qty} at {time}.

Stored for traceability or recovery.

Approach:

Framework & Tools Used:

FastAPI for building async APIs.

SQLAlchemy for DB ORM (with master-replica read/write split).

asyncpg for fast asynchronous DB updates.

RabbitMQ + aio_pika for async messaging.

Redis for caching stock values.

Prometheus for monitoring.

Rate Limiting with fastapi-limiter.

Celery as an optional worker for heavy background tasks.

Core Functionality:

POST /update-stock/:

Sends update message to RabbitMQ.

Async DB update with asyncpg.

GET /products/:

Fetches from replica DB.

POST /add-product/:

Writes to master DB.

GET /get-stock/{id}:

Checks Redis cache, falls back to DB if not cached.

Optional: Celery task for scalable stock updates

Testing Support:

- All endpoints are RESTful, easily testable via Postman.
- Use Prometheus /metrics for performance testing.
- RabbitMQ & Redis can be validated via logs and queues/cache behavior.

API Testing & Validation:

FastAPI Endpoints (with Postman)

Update stock

POST /update-stock/?product_id=1&quantity=10

Get stock (cached)

GET /get-stock/1

Add product

POST /add-product/

Body: { "name": "Shampoo", "description": "Hair care", "price": 50 }

Get product list

GET /products/

Test rate limit

Hit GET /some-data/ more than 5 times in 60s → should block.

-----THE END -----