

## RUNNING THE APPLICATION

### Step 1: Start the Server

```
python firehose_collector.py
```

```
=====
HIGH-CONCURRENCY INVENTORY SYSTEM
=====

📦 Initial Stock: 100 units
🔒 Concurrency Control: Database Row Locking
⚡ Lock Timeout: 5 seconds
=====

✅ Database initialized
📦 Initial stock: 100 units
=====

INFO: Started server process [21768]
INFO: Waiting for application startup.
=====

✅ Database initialized
📦 Initial stock: 100 units
=====

INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
|
```

### Step 2: Test with Your Browser

Test 1: Check API is alive

<http://localhost:8000>

```
{
  "service": "High-Concurrency Inventory System",
  "version": "1.0.0",
  "endpoints": {
    "POST /buy_ticket": "Purchase a ticket",
    "GET /inventory": "Check inventory status",
    "POST /reset": "Reset inventory (testing only)",
    "GET /docs": "Interactive API documentation"
  },
  "Concurrency Model": "Database row-level locking (SELECT FOR UPDATE)",
  "guarantees": [
    "Zero overselling (inventory never negative)",
    "Zero underselling (no deadlocks)",
    "Works across multiple processes"
  ]
}
```

## Test 2: Check inventory

In browser: <http://localhost:8000/inventory>

```
{  
    "item_id": 1,  
    "item_name": "Item A - Concert Ticket",  
    "current_stock": 100,  
    "initial_stock": 100,  
    "total_purchases": 0,  
    "last_updated": "2026-01-15 12:30:27"  
}
```

## RUNNING THE PROOF OF CORRECTNESS

```
C:\Users\SARVAGYA SANJAY\Desktop\InventorySystem>python proof_of_correctness.py  
=====  
▣ INVENTORY SYSTEM - PROOF OF CORRECTNESS  
=====  
  
This script proves that the inventory system:  
1. Prevents overselling (no negative inventory)  
2. Prevents underselling (no deadlocks)  
3. Works correctly with high concurrency  
4. Works across multiple processes  
  
⌚ Checking if server is running...  
Server is running!  
  
=====  
TEST 1: BASIC CONCURRENCY (100 buyers, 100 tickets)  
=====  
Inventory reset to 100 units  
Launching 100 concurrent purchase attempts...  
  
RESULTS:  
    Total time: 4.25s  
    Successful purchases: 100  
    Sold out responses: 0  
    Errors: 0  
    Average response time: 2059.84ms  
  
📦 FINAL INVENTORY:  
    Current stock: 0  
    Total purchases: 100  
  
VERIFICATION:  
    ✓ All 100 purchases succeeded  
    ✓ Final inventory is 0  
    ✓ Database has exactly 100 purchase records  
  
TEST 1 PASSED!
```

```
=====
TEST 2: OVERSELLING PREVENTION (1000 buyers, 100 tickets)
=====

Inventory reset to 100 units
Launching 1000 concurrent purchase attempts...
  (This simulates a flash sale with high contention)
  Progress: 100/1000
  Progress: 200/1000
  Progress: 300/1000
  Progress: 400/1000
  Progress: 500/1000
  Progress: 600/1000
  Progress: 700/1000
  Progress: 800/1000
  Progress: 900/1000
  Progress: 1000/1000

RESULTS:
  Total time: 41.26s
  Throughput: 24.24 requests/second

  Successful purchases: 100
  Sold out responses: 900
  Server busy responses: 0
  Errors: 0

  Response times:
    Average: 2052.64ms
    Min: 2022.03ms
    Max: 2099.55ms

FINAL INVENTORY:
  Current stock: 0
  Total purchases in DB: 100

VERIFICATION:
  ✓ Exactly 100 purchases succeeded
  ✓ All 1000 requests accounted for
  ✓ Final inventory is 0 (not negative!)
  ✓ Database has exactly 100 purchase records
  ✓ NO OVERSELLING (inventory never went negative)
```

TEST 2 PASSED! No race conditions detected!

```
=====
TEST 3: MULTIPLE PROCESSES (4 processes, 250 attempts each)
=====

Inventory reset to 100 units
Launching 4 processes...
  Each process will attempt 250 purchases
```

### TEST SUMMARY

Basic Concurrency, Overselling Prevention, and Multiple Processes: PASSED

The inventory system is CORRECT:

Zero overselling (inventory never negative)

Zero underselling (no deadlocks)

Thread-safe under high concurrency

Process-safe across multiple servers

The system is production-ready for flash sales.