UNIVERSITY OF RWANDA

African Centre of Excellence in Data Science

Cloud computing and web development

Final Exam Project Report

Group 10

NSHUNGUYIMFURA Abou-Bakar : 223028093

UWIMBABAZI Rosine : 213001787

NSABIYERA Enock : 21300178

December 2024

1. Introduction

The Inventory Management System is a web application developed in Django to manage inventory, streamline transactions, and offer real-time analytics.

By integrating cloud services and big data tools like Hadoop and Kafka, the system aims to enhance scalability, data processing efficiency, and real-time responsiveness.

2. Case Study and Objectives

Case Study: The Inventory Management System designed for a retail business struggling with manual inventory tracking and data inconsistencies.

Problems included:

- Difficulty monitoring stock levels.
- Inefficient transaction logging.
- Lack of real-time insights into sales trends and profits.

Objectives:

- Develop a web-based inventory management system.
- Integrate an authentication system for secure user access.
- Provide CRUD functionality for managing inventory and transactions.
- Build a dashboard for analytics and reporting.
- Leverage AWS MySQL database for reliable storage and accessibility.
- Use Hadoop for batch processing of large inventory datasets.
- Implement Apache Kafka for real-time data streaming and updates.

3. Web Application Development

Framework:

- Backend: Django (Python) for server-side logic and ORM integration.
- Frontend: HTML, CSS, and JavaScript for responsive and interactive user interfaces.

Features:

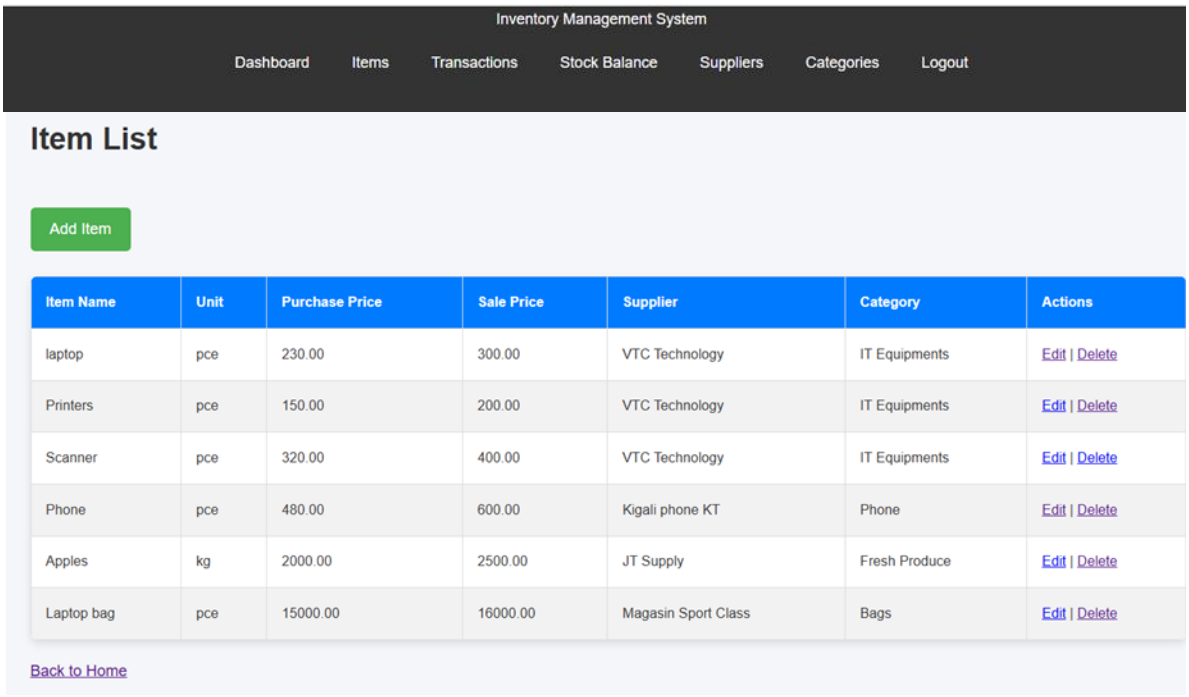
Authentication System:

- Secure login functionality using Django's built-in authentication system.
- Validation to prevent unauthorized access to sensitive data and features.

CRUD Operations:

Manage inventory data:

- **Create:** Add new items to the inventory.
- **Read:** View all items with details like stock balance, purchase price, and sale price.
- **Update:** Edit item details (e.g., prices, quantities).
- **Delete:** Remove obsolete items from the database.



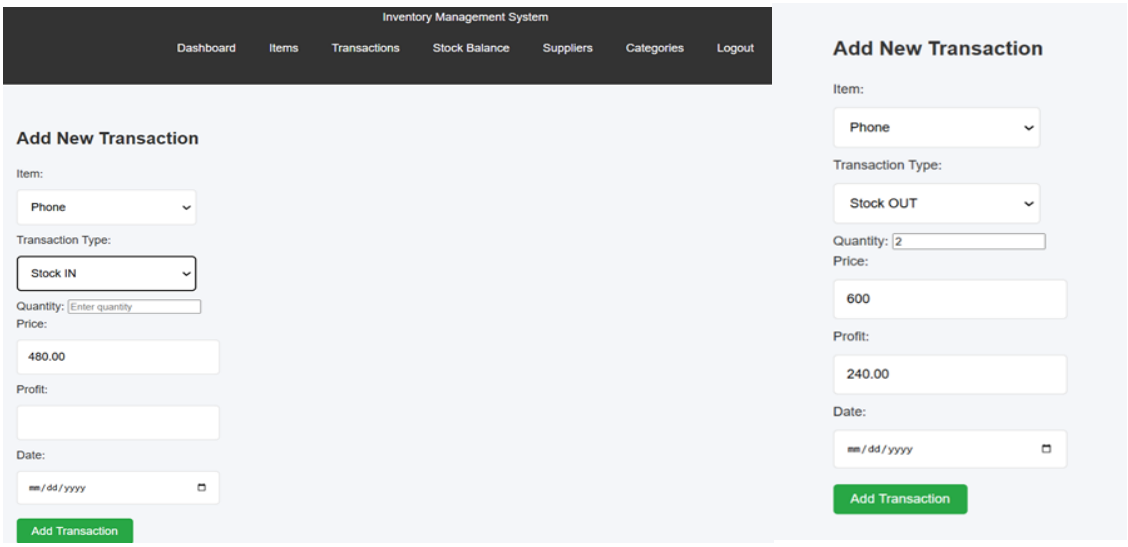
Dashboard:

Display meaningful analytics such as:

- Total Purchase Value
- Total Sales Value
- Total Profit
- Top-selling items.
- Date selector to view weekly and monthly figures.
- Real-time updates using Kafka integration.

Dynamic Forms:

Update fields like price and profit dynamically based on user input and item selection.

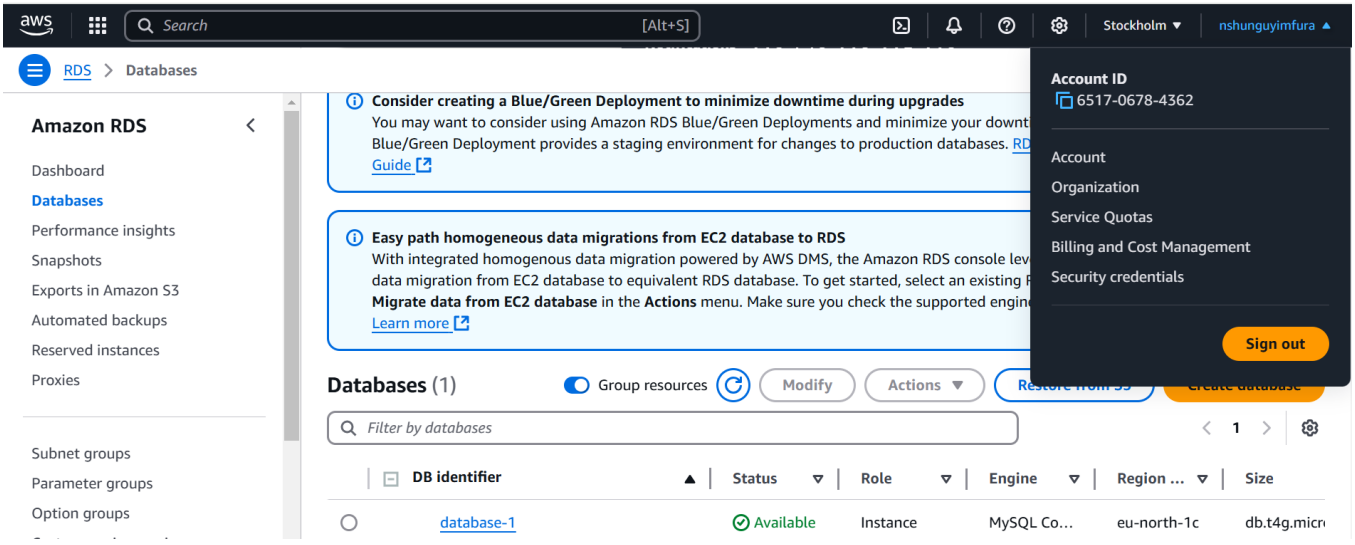


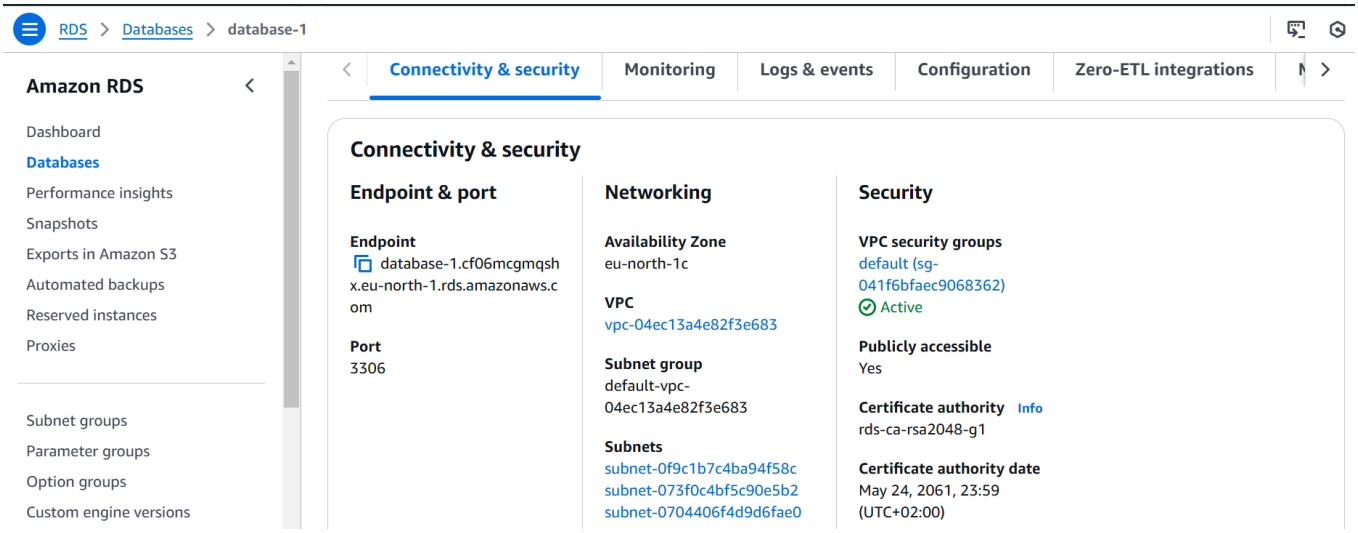
4. **Cloud Computing Integration**

MySQL Database on AWS

Amazon RDS:

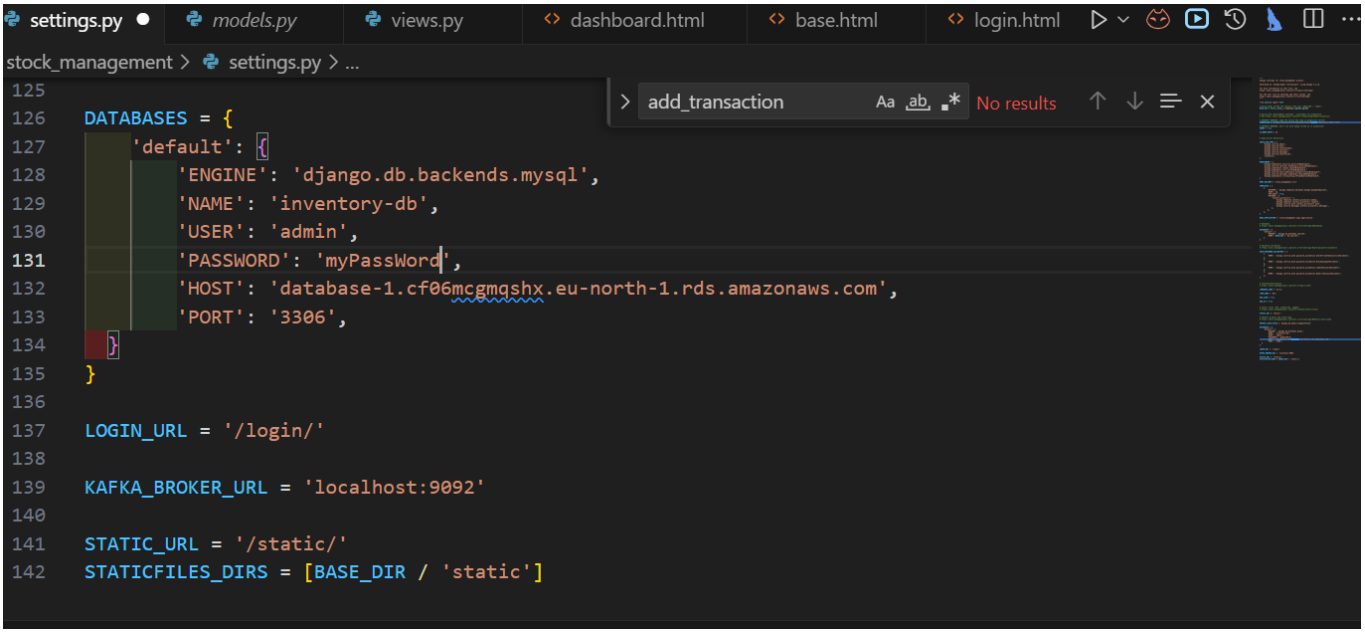
- A MySQL database was created and configured using Amazon RDS for robust, scalable data storage.
- Multi-AZ deployment was set up for high availability.
- VPC was configuration





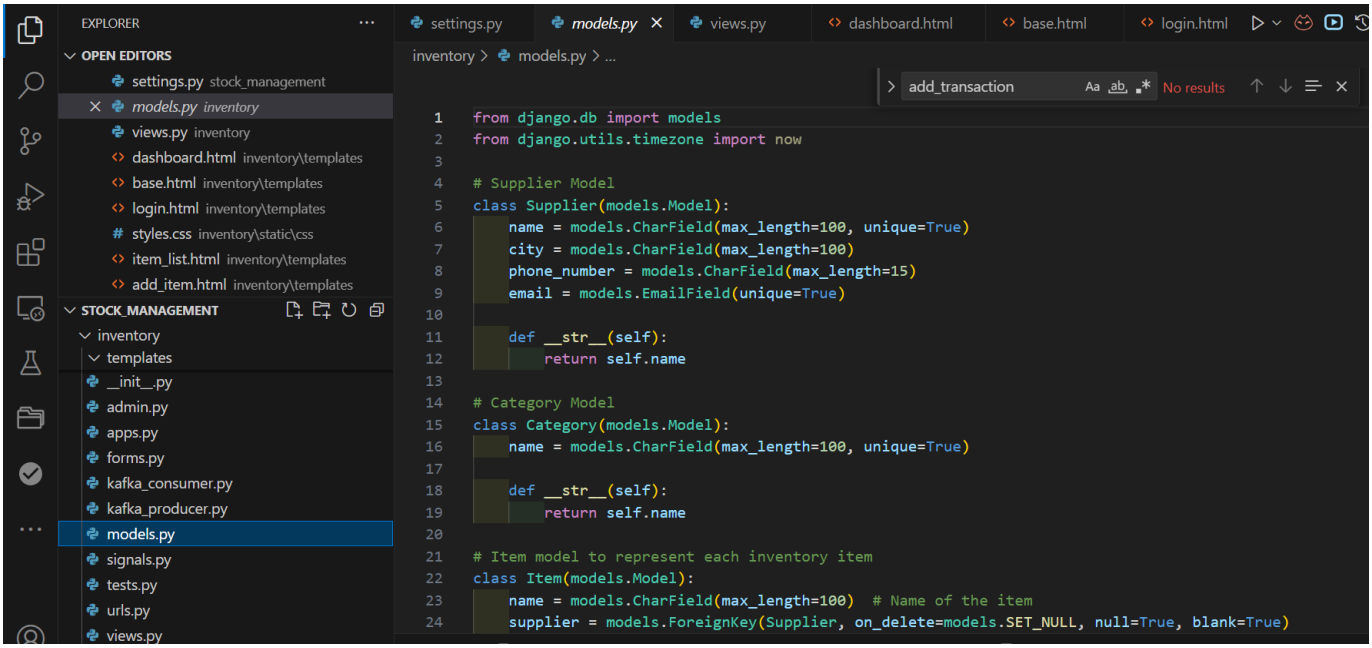
Django Integration:

- The Django application was connected to the RDS MySQL database using proper credentials and environment variables.
- The settings.py file was configured for the production database:



Transaction Handling:

- Implemented atomic transactions to ensure data consistency for stock-in and stock-out operations.
- Used Django ORM for efficient query execution.



5. Hadoop Integration

Local Hadoop Setup:

- A single-node Hadoop cluster was set up locally for batch processing.
- HDFS (Hadoop Distributed File System) was used to store transaction datasets.

MapReduce Workflow:

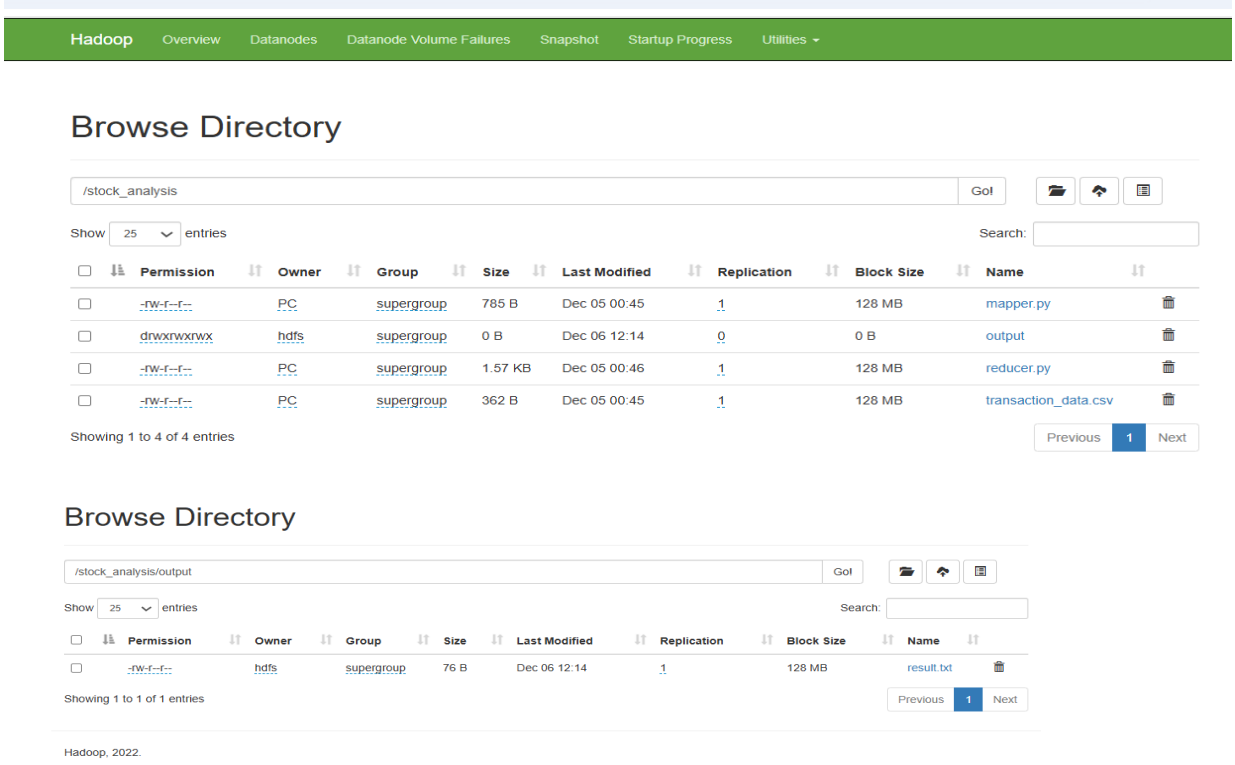
A MapReduce job was implemented to process sales data and calculate:

- Total sales for each item.
- Total profit for each item.

The following is how workflow was performed:

- Input file uploaded to HDFS: transaction_data.csv.
- Mapper processed sales records and produced intermediate key-value pairs.
- Reducer aggregated results to calculate totals.

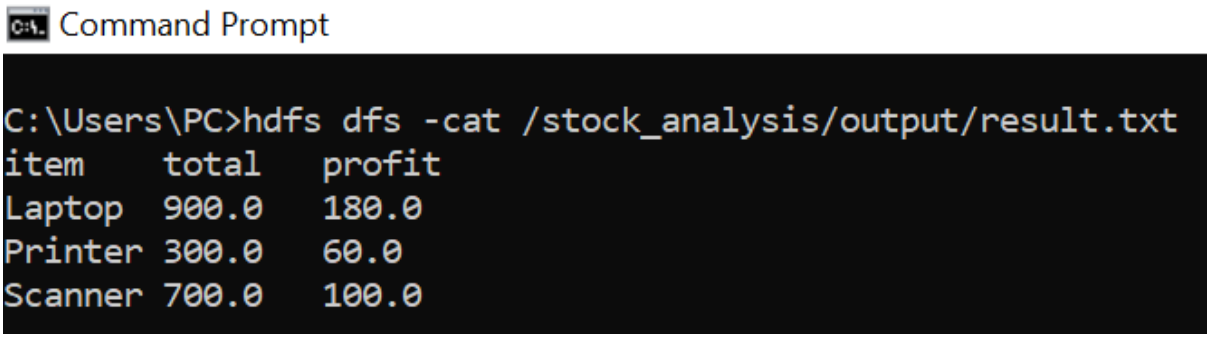
```
C:\Users\PC>hdfs dfs -mkdir /stock_analysis

C:\Users\PC>hdfs dfs -put C:\Users\PC\Desktop\Vm\transaction_data.csv /stock_analysis

C:\Users\PC>hdfs dfs -put C:\Users\PC\Desktop\Vm\mapper.py /stock_analysis

C:\Users\PC>hdfs dfs -put C:\Users\PC\Desktop\Vm\reducer.py /stock_analysis
```



Integration with Django:

- Results of the MapReduce job were displayed on the dashboard for analytics.
- Used Django views to fetch processed data from HDFS and render it in the web application.

The following is the result of the MapReduce job:



View codes for displaying results from HDFS on the web application dashboard.

```
from django.db.models import Sum, F, Q, Value
from django.db.models.functions import Coalesce  # Import Coalesce
from datetime import datetime
from hdfs import InsecureClient
from collections import defaultdict
# HDFS Configuration
HDFS_URL = 'http://localhost:9870'
HDFS_USER = 'hdfs'
OUTPUT_FILE = '/stock_analysis/output/result.txt'

client = InsecureClient(HDFS_URL, user=HDFS_USER)

@login_required
def dashboard(request):
# Fetch MapReduce results from HDFS
mapreduce_results = []
try:
    with client.read(OUTPUT_FILE, encoding='utf-8') as reader:
        lines = reader.readlines()[1:]  # Skip the header line
        for line in lines:
            item, total, profit = line.strip().split("\t")
            mapreduce_results.append({
                "item": item,
                "total": float(total),
                "profit": float(profit),
            })
```

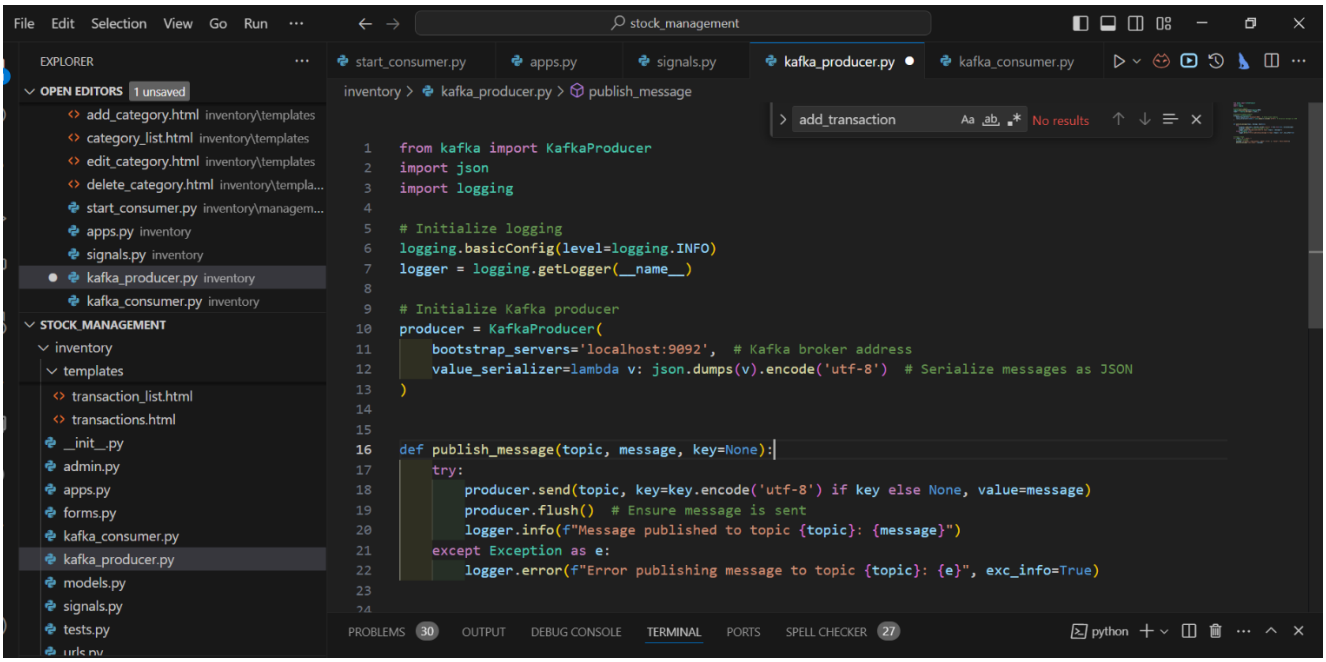
6. Apache Kafka Integration

Setup and Configuration:

- Kafka was set up locally to handle real-time data streaming.
- Created a Kafka topic for publishing stock transactions.

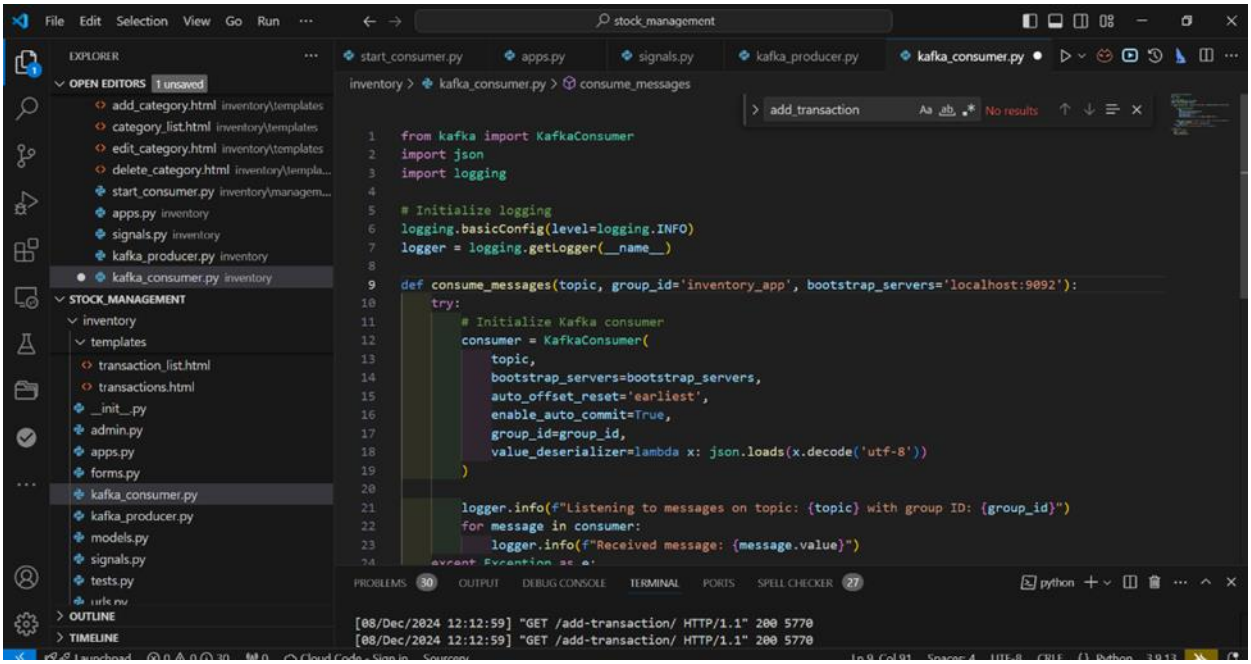
Data Publishing:

- Stock transactions (e.g., stock-in and stock-out) were published as messages to the Kafka topic:



Data Consumption:

- A Kafka consumer was implemented to consume messages from a Kafka topic and update the dashboard in real-time:



7. **Challenges and Solutions**

Database Connectivity Issues:

- **Challenge:** Connection errors during AWS MySQL integration.
- **Solution:** Verified RDS endpoint, security groups, and IAM roles to ensure proper access.

Real-Time Streaming Implementation:

- **Challenge:** Setting up Kafka and integrating with the application.
- **Solution:** Used Python's Kafka library and debugged network issues with Zookeeper.

Hadoop Integration:

- **Challenge:** Processing datasets efficiently.
- **Solution:** Optimized MapReduce logic and ensured proper HDFS configurations.

8. **Conclusion**

The Inventory Management System successfully achieves its objectives by providing a secure, scalable, and efficient solution for inventory management. The integration with AWS ensures high availability, and the implemented features address the critical requirements of stock validation, profit calculation, and user-friendly interactions.

The challenges faced during development were resolved with thoughtful design decisions and robust technical implementations, resulting in a highly functional and user-focused application.

End!!!!