**Documentation: Flask Application for Supply Chain Dashboard**

**Overview**

This Flask application serves as a real-time supply chain risk dashboard. It integrates with a MySQL database to fetch and display data related to inventory adjustments and supply chain risks. The dashboard is designed to provide insights into supply chain disruptions through data visualizations and email notifications.

**Features**

1. **Real-Time Risk Data Visualization**
   * Displays risk levels distribution across the supply chain.
   * Shows inventory adjustments made across various countries.
2. **Email Notifications**
   * Sends email alerts for critical risk levels (High/Low) and significant inventory adjustments.
3. **Dashboard Rendering**
   * Displays dynamic visualizations on the dashboard using matplotlib.
   * Renders the dashboard HTML template to present the data and visualizations.

**Components**

**1. Flask Application (main.py)**

The main Flask app serves as the backbone of the application, handling routes, database connections, and rendering visualizations. Below is an analysis of the key sections of the code:

**Flask Application Setup**

python

from flask import Flask, render\_template, request

import mysql.connector

import pandas as pd

import matplotlib.pyplot as plt

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

import os

app = Flask(\_\_name\_\_) # Flask application initialization

* **Flask** is imported for web routing and template rendering.
* **mysql.connector**: MySQL connector to interact with the database.
* **pandas**: Data analysis library to manage and process data.
* **matplotlib**: Used for generating visualizations (bar charts).
* **smtplib** and **email.mime**: Used for sending email notifications.

**Database Connection (connect\_to\_db function)**

python

def connect\_to\_db():

return mysql.connector.connect(

host="127.0.0.1",

user="root",

password="Omkar@003",

database="inventory\_management"

)

* Connects to a local MySQL database named inventory\_management using specified credentials.

**Email Configuration and Sending Function**

python

SMTP\_SERVER = "smtp.gmail.com"

SMTP\_PORT = 587

SENDER\_EMAIL = "sender@gmail.com"

SENDER\_PASSWORD = "password" # Use an app-specific password

RECIPIENT\_EMAIL = "recipent@gmail.com"

def send\_email(subject, body):

try:

msg = MIMEMultipart()

msg["From"] = SENDER\_EMAIL

msg["To"] = RECIPIENT\_EMAIL

msg["Subject"] = subject

msg.attach(MIMEText(body, "plain"))

server = smtplib.SMTP(SMTP\_SERVER, SMTP\_PORT)

server.starttls()

server.login(SENDER\_EMAIL, SENDER\_PASSWORD)

server.sendmail(SENDER\_EMAIL, RECIPIENT\_EMAIL, msg.as\_string())

server.quit()

print(f"Email sent: {subject}")

except Exception as e:

print(f"Failed to send email: {e}")

* **SMTP configuration**: Gmail SMTP settings are used for sending email notifications.
* **send\_email function**: Sends an email with the subject and body specified. It connects to the SMTP server, logs in, and sends the email.

**Dashboard Route (/dashboard)**

python

@app.route("/dashboard")

def dashboard():

conn = connect\_to\_db()

cursor = conn.cursor()

try:

cursor.execute("SELECT \* FROM adjusted\_inventory")

adjusted\_inventory = pd.DataFrame(cursor.fetchall(), columns=["Country", "Stock Level", "Stock Adjusted", "Adjustment", "Published At"])

cursor.execute("SELECT \* FROM risk\_data")

risk\_data = pd.DataFrame(cursor.fetchall(), columns=["ID", "Title", "Risk Level", "Stock Adjustment", "Published At"])

# Create visualizations

# Risk Levels Distribution

risk\_level\_counts = risk\_data["Risk Level"].value\_counts()

plt.figure(figsize=(8, 6))

risk\_level\_counts.plot(kind="bar", color=["red", "orange", "green"])

plt.title("Risk Levels Distribution")

plt.xlabel("Risk Level")

plt.ylabel("Count")

plt.savefig("static/risk\_levels.png")

plt.close()

# Inventory Adjustments by Country

plt.figure(figsize=(10, 6))

adjusted\_inventory.set\_index("Country")["Adjustment"].plot(kind="bar", color="blue")

plt.title("Inventory Adjustments by Country")

plt.xlabel("Country")

plt.ylabel("Adjustment")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.savefig("static/inventory\_adjustments.png")

plt.close()

# Trigger email notifications

critical\_risks = risk\_data[risk\_data["Risk Level"].isin(["High", "Low"])]

for \_, row in critical\_risks.iterrows():

subject = f"Risk Alert: {row['Title']}"

body = f"Risk Level: {row['Risk Level']}\nTitle: {row['Title']}\nStock Adjustment: {row['Stock Adjustment']}\nPublished At: {row['Published At']}"

send\_email(subject, body)

for \_, row in adjusted\_inventory.iterrows():

subject = f"Stock Adjustment Alert: {row['Country']}"

body = f"Country: {row['Country']}\nStock Level: {row['Stock Level']}\nStock Adjusted: {row['Stock Adjusted']}\nAdjustment: {row['Adjustment']}\nPublished At: {row['Published At']}"

send\_email(subject, body)

return render\_template("dashboard.html", risk\_image="risk\_levels.png", inventory\_image="inventory\_adjustments.png")

except Exception as e:

return f"Error: {e}"

finally:

cursor.close()

conn.close()

* **Database Fetching**: The database is queried for adjusted\_inventory and risk\_data tables.
* **Data Processing**: The data is processed and visualized using matplotlib.
* **Visualizations**: Bar charts are generated and saved as images in the static directory (risk\_levels.png, inventory\_adjustments.png).
* **Email Alerts**: Critical risk levels and significant inventory adjustments trigger email notifications.

**Run Flask Application**

python

if \_\_name\_\_ == "\_\_main\_\_":

if not os.path.exists("static"):

os.mkdir("static") # Ensure static directory exists for saving images

app.run(debug=True)

* Starts the Flask app and ensures that the static directory exists for storing images.

**HTML Template (dashboard.html)**

This HTML template displays the visualizations generated by the Flask app.

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Supply Chain Dashboard</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

}

h1 {

color: #2c3e50;

}

h2 {

color: #34495e;

}

img {

width: 80%;

margin-bottom: 20px;

}

</style>

</head>

<body>

<h1>Supply Chain Risk Dashboard</h1>

<h2>Risk Levels Distribution</h2>

<img src="{{ url\_for('static', filename=risk\_image) }}" alt="Risk Levels">

<h2>Inventory Adjustments by Country</h2>

<img src="{{ url\_for('static', filename=inventory\_image) }}" alt="Inventory Adjustments">

</body>

</html>

* **Visualizations**: Displays two images (risk\_levels.png and inventory\_adjustments.png) generated by the Flask app.
* **Layout**: Simple and clean layout with h1 and h2 headings, and images styled to occupy 80% of the page width.

**Key Dependencies**

1. **Flask**: Web framework for routing and rendering templates.
2. **mysql.connector**: MySQL database connector for querying data.
3. **pandas**: Data handling and manipulation.
4. **matplotlib**: For generating charts and visualizations.
5. **smtplib**: For sending email notifications.

**Deployment Notes**

* Ensure that the Flask app is running on a server that has access to the MySQL database.
* Replace SMTP credentials with your own (if using Gmail, consider using an app-specific password for enhanced security).
* Modify the RECIPIENT\_EMAIL to the desired address for receiving email alerts.