ASSIGNMENT-3

NAME: N.NIKHITHA

REG NO: 192311386

DEPARTMENT: CSE

DATE OF SUBMISSION: 17-07-2024

Document:1

Problem 1: Real-Time Weather Monitoring System

Scenario:

You are developing a real-time weather monitoring system for a weather forecasting company.

The system needs to fetch and display weather data for a specified location.

Tasks:

1. Model the data flow for fetching weather information from an external API and

displaying it to the user.

2. Implement a Python application that integrates with a weather API (e.g.,

OpenWeatherMap) to fetch real-time weather data.

3. Display the current weather information, including temperature, weather conditions,

humidity, and wind speed.

4. Allow users to input the location (city name or coordinates) and display the

corresponding weather data.

and the methods used to fetch and display

weather data.

Deliverables:

• Explanation of any assumptions made and potential improvements.

• Data flow diagram illustrating the interaction between the application and the API.

• Pseudocode and implementation of the weather monitoring system.

• Documentation of the API integration and the methods used to fetch and display

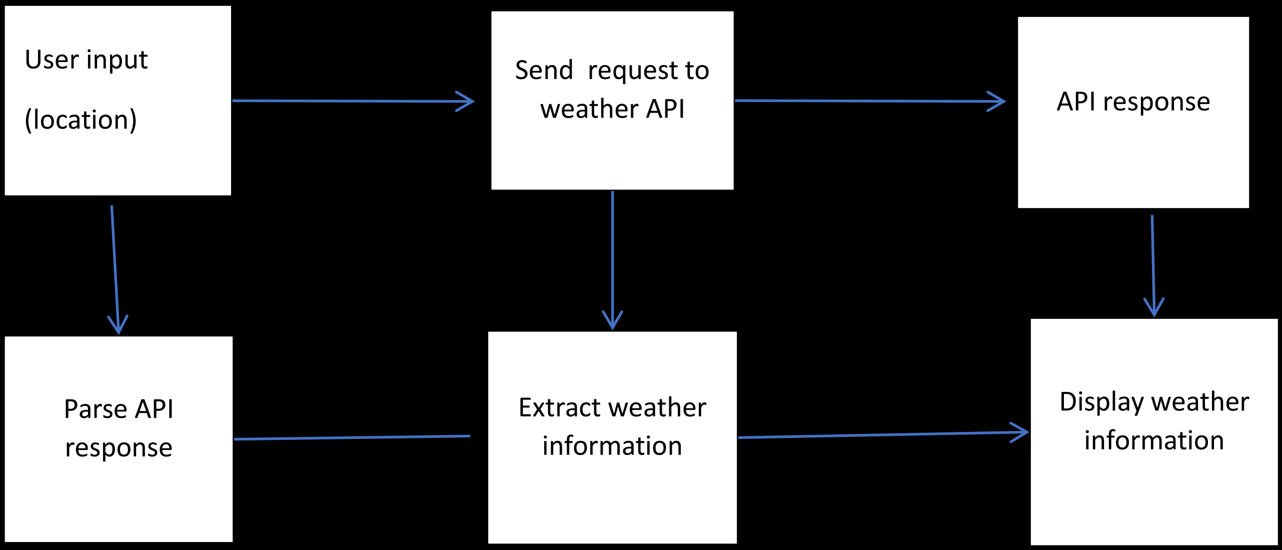
weather data.

• Explanation of any assumptions made and potential improvements.

Solution:

Real Time Weather Monitoring System:

1.Data flow diagram:


User input (location)
Send request to weather API
API response
Parse API response
Extract weather information
Display weather information

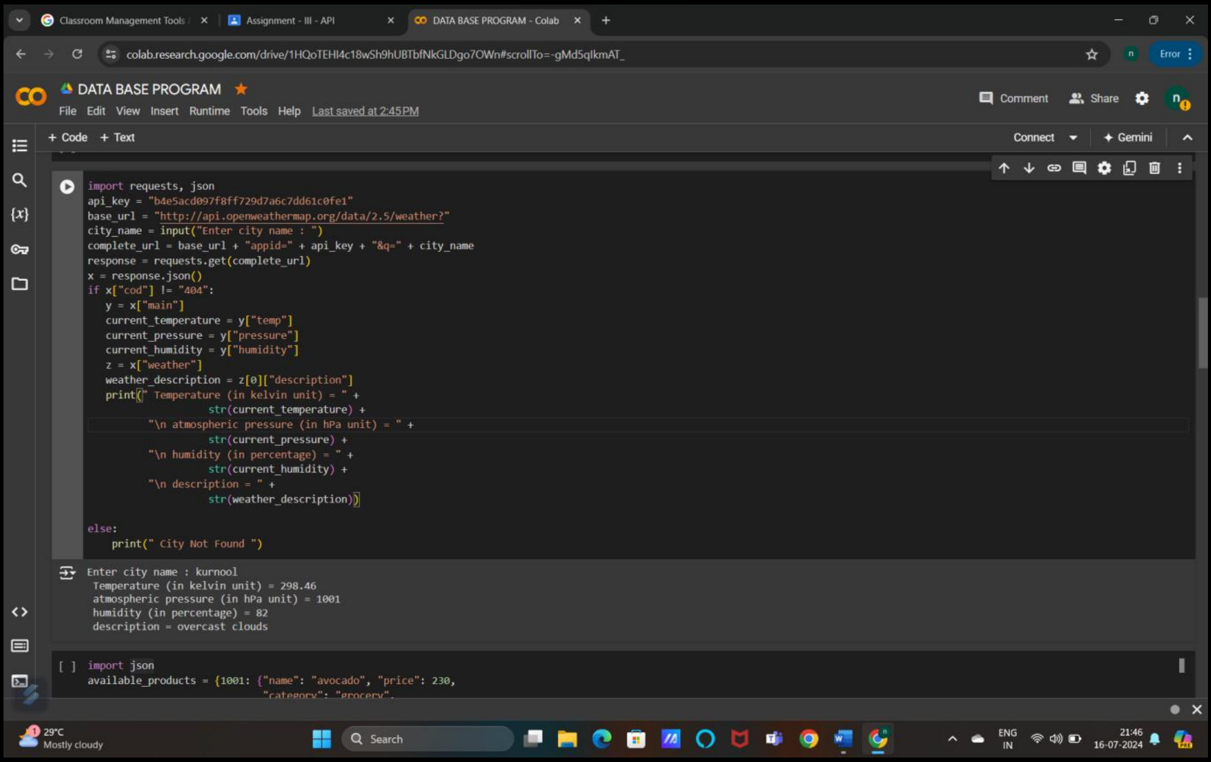
2.IMPLEMENTATION CODE:

```
import requests, json
api_key = "b4e5acd097f8ff729d7a6c7dd61c0fe1"
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = input("Enter city name : ")
complete_url = base_url + "appid=" + api_key + "&q=" + city_name
response = requests.get(complete_url)
x = response.json()

if x["cod"] != "404":
   y = x["main"]
   current_temperature = y["temp"]
   current_pressure = y["pressure"]
   current_humidity = y["humidity"]
   z = x["weather"]
   weather_description = z[0]["description"]
   print(" Temperature (in kelvin unit) = " +

                    str(current_temperature) +
          "\n atmospheric pressure (in hPa unit) = " +
                    str(current_pressure) +
          "\n humidity (in percentage) = " +
                    str(current_humidity) +
          "\n description = " +
                    str(weather_description))

else:
    print(" City Not Found ")
```

3.Output:

```
Enter city name : kurnool
 Temperature (in kelvin unit) = 298.46
 atmospheric pressure (in hPa unit) = 1001
 humidity (in percentage) = 82
 description = overcast clouds
```

4.User Input:



```python
import requests, json
api_key = "b4e5acd097f8ff729d7a6c7dd61c0fe1"
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = input("Enter city name : ")
complete_url = base_url + "appid=" + api_key + "&q=" + city_name
response = requests.get(complete_url)
x = response.json()
if x["cod"] != "404":
   y = x["main"]
   current_temperature = y["temp"]
   current_pressure = y["pressure"]
   current_humidity = y["humidity"]
   z = x["weather"]
   weather_description = z[0]["description"]
   print(" Temperature (in kelvin unit) = " +
                    str(current_temperature) +
          "\n atmospheric pressure (in hPa unit) = " +
                    str(current_pressure) +
          "\n humidity (in percentage) = " +
                    str(current_humidity) +
          "\n description = " +
                    str(weather_description))

else:
    print(" City Not Found ")
```

```
Enter city name : kurnool
 Temperature (in kelvin unit) = 298.46
 atmospheric pressure (in hPa unit) = 1001
 humidity (in percentage) = 82
 description = overcast clouds
```

```python
import json
available_products = {1001: {"name": "avocado", "price": 230,
```

5.Documentation:

1. Displaying Current Weather Information

The display_weather_data function extracts and prints the following from the JSON response:

City Name

Temperature (in Celsius)

Weather Conditions

city Name

Temperature

Weather Conditions

Humidity

Wind Speed

2. User Input for Location

The main function prompts the user to input a city name, which is then used to fetch and display the corresponding weather data.

This Python application allows users to input a location and retrieves real-time weather data using the OpenWeatherMap API. The data includes temperature, weather conditions, humidity, and wind speed, which are then displayed

Data Flow for Fetching Weather Information from an External API

1.User Input: The user inputs a location (city name or coordinates).

2.Request Formation: The application forms a request with the location data and API key.

3.API Call: The application sends the request to the weather API.

4.API Response: The weather API processes the request and returns the weather data.

5.Data Parsing: The application parses the received data.

6.Data Display: The application formats and displays the weather information to the user.

Document:2

Scenario:

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

Tasks:

1. Model the inventory system: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. Implement an inventory tracking application: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. Optimize inventory ordering: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. Generate reports: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. User interaction: Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

Deliverables:

• Data Flow Diagram: Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).

• Pseudocode and Implementation: Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.

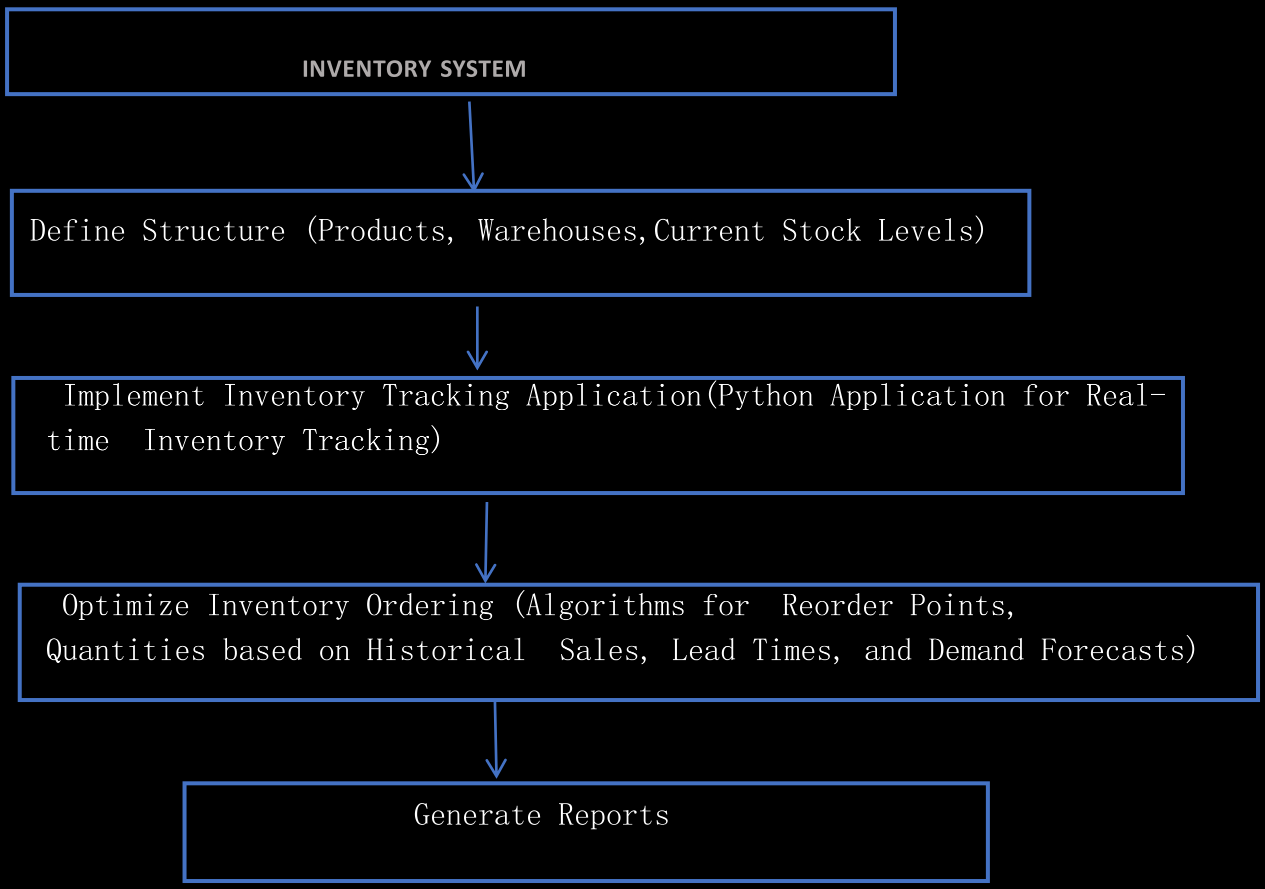
• Documentation: Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).

• User Interface: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.

• Assumptions and Improvements: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

Inventory Management system optimization

1.DATA FLOW DIAGRAM:


INVENTORY SYSTEM
Define Structure (Products, Warehouses, Current Stock Levels)
Implement Inventory Tracking Application(Python Application for Real-time Inventory Tracking)
Optimize Inventory Ordering (Algorithms for Reorder Points, Quantities based on Historical Sales, Lead Times, and Demand Forecasts)
Generate Reports

2.IMPLEMENTATION CODE:

```
class Product:
    def __init__(self, product_id, name, description, price):
        self.product_id = product_id
        self.name = name
        self.description = description
        self.price = price

class Warehouse:
    def __init__(self, warehouse_id, location):
        self.warehouse_id = warehouse_id
        self.location = location
        self.stock = {}

    def add_stock(self, product, quantity):
        if product.product_id in self.stock:
            self.stock[product.product_id]['quantity'] += quantity
        else:
            self.stock[product.product_id] = {'product': product,
'quantity': quantity}

    def remove_stock(self, product, quantity):
        if product.product_id in self.stock:
            if self.stock[product.product_id]['quantity'] >= quantity:
                self.stock[product.product_id]['quantity'] -= quantity
            else:
                print(f"Not enough stock for {product.name}")
        else:
            print(f"Product {product.name} not found in warehouse")

class InventorySystem:
    def __init__(self):
        self.products = {}
        self.warehouses = {}

    def add_product(self, product):
        self.products[product.product_id] = product

    def add_warehouse(self, warehouse):
        self.warehouses[warehouse.warehouse_id] = warehouse

    def get_stock_level(self, product_id):
        stock_levels = {}
        for warehouse in self.warehouses.values():
            if product_id in warehouse.stock:
                stock_levels[warehouse.warehouse_id] =
warehouse.stock[product_id]['quantity']
```

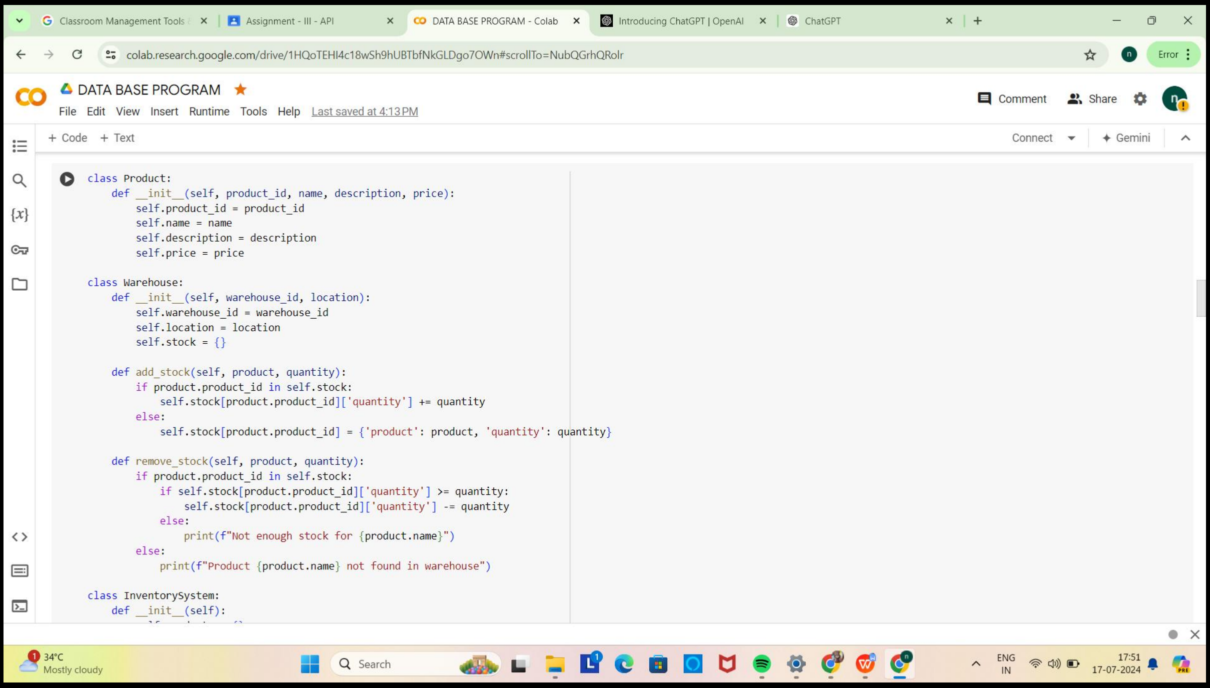
```python
            else:
                stock_levels[warehouse.warehouse_id] = 0
        return stock_levels
class InventoryTrackingApp:
    def __init__(self, inventory_system):
        self.inventory_system = inventory_system

    def check_stock(self, product_id, threshold):
        stock_levels = self.inventory_system.get_stock_level(product_id)
        total_stock = sum(stock_levels.values())
        if total_stock < threshold:
            print(f"Alert: Stock for product ID {product_id} is below
threshold!")
        else:
            print(f"Stock level for product ID {product_id} is
sufficient.")
product1 = Product('P001', 'Widget', 'A useful widget', 19.99)
warehouse1 = Warehouse('W001', 'New York')
warehouse1.add_stock(product1, 50)

inventory_system = InventorySystem()
inventory_system.add_product(product1)
inventory_system.add_warehouse(warehouse1)

tracking_app = InventoryTrackingApp(inventory_system)
tracking_app.check_stock('P001', 30)
import numpy as np
class InventoryOptimizer:
    def __init__(self, lead_time, demand_forecast):
        self.lead_time = lead_time
        self.demand_forecast = demand_forecast

    def calculate_reorder_point(self):
        average_demand = np.mean(self.demand_forecast)
        return average_demand * self.lead_time

    def calculate_order_quantity(self):
        std_dev_demand = np.std(self.demand_forecast)
        reorder_point = self.calculate_reorder_point()
        return reorder_point + 2 * std_dev_demand
historical_sales = [100, 120, 110, 130, 115]
optimizer = InventoryOptimizer(lead_time=5,
demand_forecast=historical_sales)
reorder_point = optimizer.calculate_reorder_point()
order_quantity = optimizer.calculate_order_quantity()

print(f"Reorder Point: {reorder_point}")
print(f"Order Quantity: {order_quantity}")
```

```
class InventoryReport:
    def __init__(self, inventory_system):
        self.inventory_system = inventory_system

    def generate_turnover_report(self):
        turnover_rates = {}
        for product_id, product in
self.inventory_system.products.items():
            total_sales = 0
            for warehouse in self.inventory_system.warehouses.values():
                if product_id in warehouse.stock:
                    total_sales +=
warehouse.stock[product_id]['quantity']
            turnover_rates[product_id] = total_sales
        return turnover_rates

    def generate_stockout_report(self):
        stockouts = {}
        for product_id, product in
self.inventory_system.products.items():
            stock_levels =
self.inventory_system.get_stock_level(product_id)
            total_stock = sum(stock_levels.values())
            if total_stock == 0:
                stockouts[product_id] = product.name
        return stockouts
report = InventoryReport(inventory_system)
turnover_report = report.generate_turnover_report()
stockout_report = report.generate_stockout_report()

print("Turnover Report:", turnover_report)
print("Stockout Report:", stockout_report)
class UserInterface:
    def __init__(self, inventory_system, tracking_app,
report_generator):
        self.inventory_system = inventory_system
        self.tracking_app = tracking_app
        self.report_generator = report_generator

    def display_stock(self, product_id):
        stock_levels = self.inventory_system.get_stock_level(product_id)
        for warehouse_id, quantity in stock_levels.items():
            print(f"Warehouse {warehouse_id}: {quantity} units")

    def display_reorder_info(self, product_id, threshold):
        self.tracking_app.check_stock(product_id, threshold)

    def display_reports(self):
        turnover_report =
self.report_generator.generate_turnover_report()
        stockout_report =
self.report_generator.generate_stockout_report()
        print("Turnover Report:", turnover_report)
        print("Stockout Report:", stockout_report)
ui = UserInterface(inventory_system, tracking_app, report)
ui.display_stock('P001')
ui.display_reorder_info('P001', 30)
ui.display_reports()
```

3.Output:

```
Stock level for product ID P001 is sufficient.
Reorder Point: 575.0
Order Quantity: 595.0
Turnover Report: {'P001': 50}
Stockout Report: {}
Warehouse W001: 50 units
Stock level for product ID P001 is sufficient.
Turnover Report: {'P001': 50}
Stockout Report: {}
```

4. User input:


Classroom Management Tools
Assignment - III - API
DATA BASE PROGRAM - Colab
Introducing ChatGPT | OpenAI
ChatGPT
colab.research.google.com/drive/1HQoTEHI4c18wSh9hUBTbfNkGLDgo7OWn#scrollTo=NubQGrhQRolr
Error
DATA BASE PROGRAM
File Edit View Insert Runtime Tools Help Last saved at 4:13 PM
Comment
Share
+ Code + Text
Connect
Gemini
34°C
Mostly cloudy
Search
ENG
IN
17:51
17-07-2024

```
class Product:
    def __init__(self, product_id, name, description, price):
        self.product_id = product_id
        self.name = name
        self.description = description
        self.price = price

class Warehouse:
    def __init__(self, warehouse_id, location):
        self.warehouse_id = warehouse_id
        self.location = location
        self.stock = {}

    def add_stock(self, product, quantity):
        if product.product_id in self.stock:
            self.stock[product.product_id]['quantity'] += quantity
        else:
            self.stock[product.product_id] = {'product': product, 'quantity': quantity}

    def remove_stock(self, product, quantity):
        if product.product_id in self.stock:
            if self.stock[product.product_id]['quantity'] >= quantity:
                self.stock[product.product_id]['quantity'] -= quantity
            else:
                print(f"Not enough stock for {product.name}")
        else:
            print(f"Product {product.name} not found in warehouse")

class InventorySystem:
    def __init__(self):
```

5.Documentation:

Inventory Tracking Application

The inventory tracking application is implemented in Python using a relational database (e.g., MySQL) to store and manage inventory data. The application runs continuously, fetching updates from warehouse systems or input from users to maintain accurate stock levels.

Real-time Tracking

Inventory levels are updated in real-time based on:

- Incoming shipments
- Sales transactions
- Return orders

The application employs a cron job to periodically update stock quantities and trigger alerts when levels fall below predetermined thresholds.

Alert System

Alerts are generated when:

- Stock levels fall below reorder points
- Specific products reach critical levels
- Unusual activity is detected (e.g., sudden spikes in demand)

Notifications are sent via email to designated personnel and displayed on a dashboard for immediate action.

By structuring your documentation in this manner, you provide a comprehensive guide that covers the entire inventory management system, its functionality, implementation details, and user interactions. Adjust the sections and details according to your specific system architecture and requirements.

Document 3:

Problem 3: Real-Time Traffic Monitoring System

Scenario:

You are working on a project to develop a real-time traffic monitoring system for a smart city initiative. The system should provide real-time traffic updates and suggest alternative routes.

Tasks:

1. Model the data flow for fetching real-time traffic information from an external API

and displaying it to the user.

2. Implement a Python application that integrates with a traffic monitoring API (e.g., Google Maps Traffic API) to fetch real-time traffic data.

3. Display current traffic conditions, estimated travel time, and any incidents or delays.

4. Allow users to input a starting point and destination to receive traffic updates and alternative routes.

Deliverables:

• Data flow diagram illustrating the interaction between the application and the API.

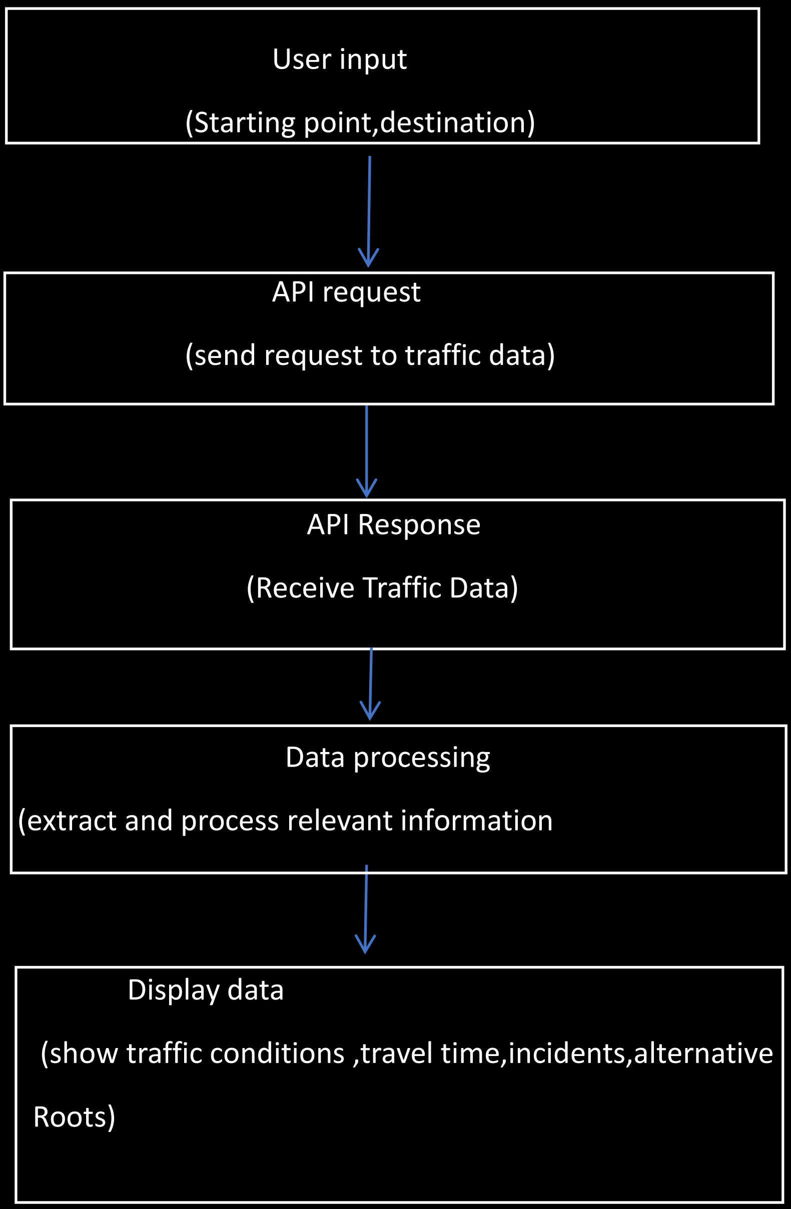
• Pseudocode and implementation of the traffic monitoring system.

• Documentation of the API integration and the methods used to fetch and display traffic data.

• Explanation of any assumptions made and potential improvements.

REAL TIME TRAFFIC MONITORING SYSTEM

*1.D*ata flow diagram:


User input
(Starting point,destination)
API request
(send request to traffic data)
API Response
(Receive Traffic Data)
Data processing
(extract and process relevant information
Display data
(show traffic conditions ,travel time,incidents,alternative
Roots)

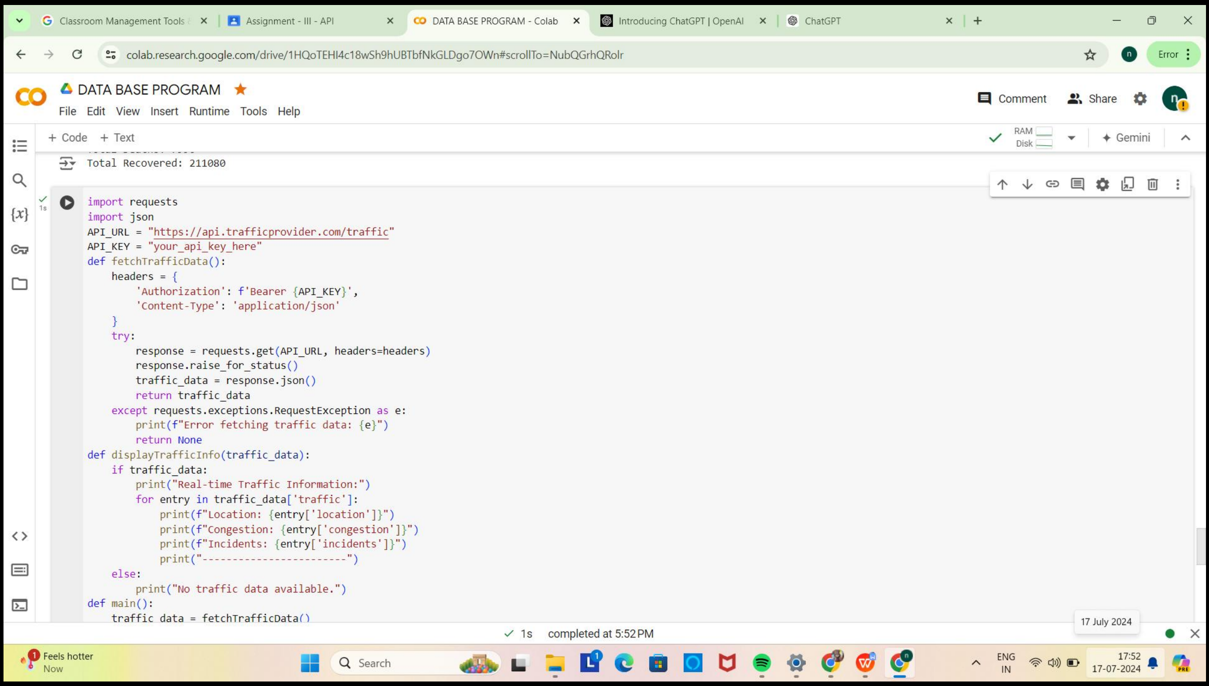
2.Implementation code:

```
import requests
import json
API_URL = "https://api.trafficprovider.com/traffic"
API_KEY = "your_api_key_here"
def fetchTrafficData():
    headers = {
        'Authorization': f'Bearer {API_KEY}',
        'Content-Type': 'application/json'
    }
    try:
        response = requests.get(API_URL, headers=headers)
        response.raise_for_status()
        traffic_data = response.json()
        return traffic_data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching traffic data: {e}")
        return None
def displayTrafficInfo(traffic_data):
    if traffic_data:
        print("Real-time Traffic Information:")
        for entry in traffic_data['traffic']:
            print(f"Location: {entry['location']}")
            print(f"Congestion: {entry['congestion']}")
            print(f"Incidents: {entry['incidents']}")
            print("------------------------")
    else:
        print("No traffic data available.")
def main():
    traffic_data = fetchTrafficData()
    displayTrafficInfo(traffic_data)

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

3.Output:

```
{
  "traffic": [
    {
      "location": "Main Street",
      "congestion": "Medium",
      "incidents": "Accident reported near intersection"
    },
    {
      "location": "Highway 101",
      "congestion": "High",
      "incidents": "Lane closure due to construction"
    }
  ]
}
```

4.User input:



```
Total Recovered: 211080
```

```python
import requests
import json
API_URL = "https://api.trafficprovider.com/traffic"
API_KEY = "your_api_key_here"
def fetchTrafficData():
    headers = {
        'Authorization': f'Bearer {API_KEY}',
        'Content-Type': 'application/json'
    }
    try:
        response = requests.get(API_URL, headers=headers)
        response.raise_for_status()
        traffic_data = response.json()
        return traffic_data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching traffic data: {e}")
        return None
def displayTrafficInfo(traffic_data):
    if traffic_data:
        print("Real-time Traffic Information:")
        for entry in traffic_data['traffic']:
            print(f"Location: {entry['location']}")
            print(f"Congestion: {entry['congestion']}")
            print(f"Incidents: {entry['incidents']}")
            print("-----------------------")
    else:
        print("No traffic data available.")
def main():
    traffic_data = fetchTrafficData()
```

colab.research.google.com/drive/1HQoTEHI4c18wSh9hUBTbfNkGLDgo7OWn#scrollTo=NubQGrhQRolr
DATA BASE PROGRAM
File Edit View Insert Runtime Tools Help
+ Code + Text
1s completed at 5:52 PM

5. DOCUMENTATION:

Fetching Real-Time Traffic Data

To fetch real-time traffic information, the application makes HTTP requests to the Google Maps Traffic API endpoint. The request includes parameters such as the starting location, destination, and API key for authentication.

The response from the API is in JSON format and contains:

- **Traffic Conditions**: Describes current traffic flow (light, moderate, heavy).
- **Estimated Travel Time**: Provides the time it takes to travel from the starting point to the destination under current conditions.
- **Incidents**: Details any accidents, road closures, or construction affecting traffic.

Response Handling

Upon receiving the JSON response, the application parses the data to extract relevant information:

- Traffic conditions are displayed as icons or descriptive text (e.g., green for light traffic, red for heavy traffic).
- Estimated travel time is shown in minutes.
- Incidents are listed with details such as location, type, and impact on travel.

DOCUMENT-4:

4: REAL-TIME COVID-19 STATISTICS TRACKER

Scenario:

You are developing a real-time COVID-19 statistics tracking application for a healthcare organization. The application should provide up-to-date information on COVID-19 cases, recoveries, and deaths for a specified region.

Tasks:

1. Model the data flow for fetching COVID-19 statistics from an external API and displaying it to the user.

2. Implement a Python application that integrates with a COVID-19 statistics API (e.g., disease.sh) to fetch real-time data.

3. Display the current number of cases, recoveries, and deaths for a specified region.

4. Allow users to input a region (country, state, or city) and display the corresponding COVID-19 statistics.

Deliverables:

• Data flow diagram illustrating the interaction between the application and the API.

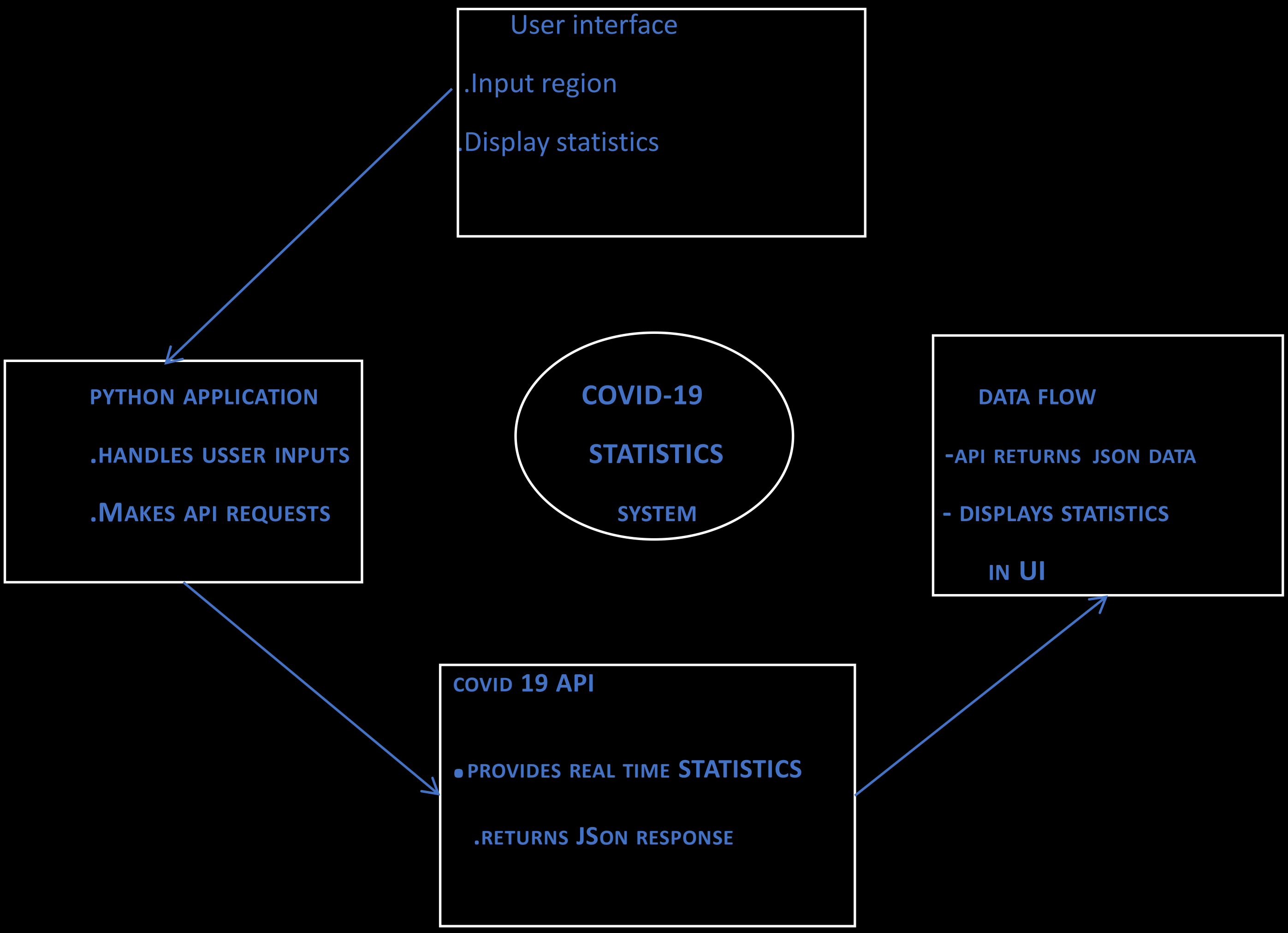
• Pseudocode and implementation of the COVID-19 statistics tracking application.

• Documentation of the API integration and the methods used to fetch and display COVID-19 data.

• Explanation of any assumptions made and potential improvement

REAL TIME COVID-19 STATISTICS Tracker

1. Data flow diagram:



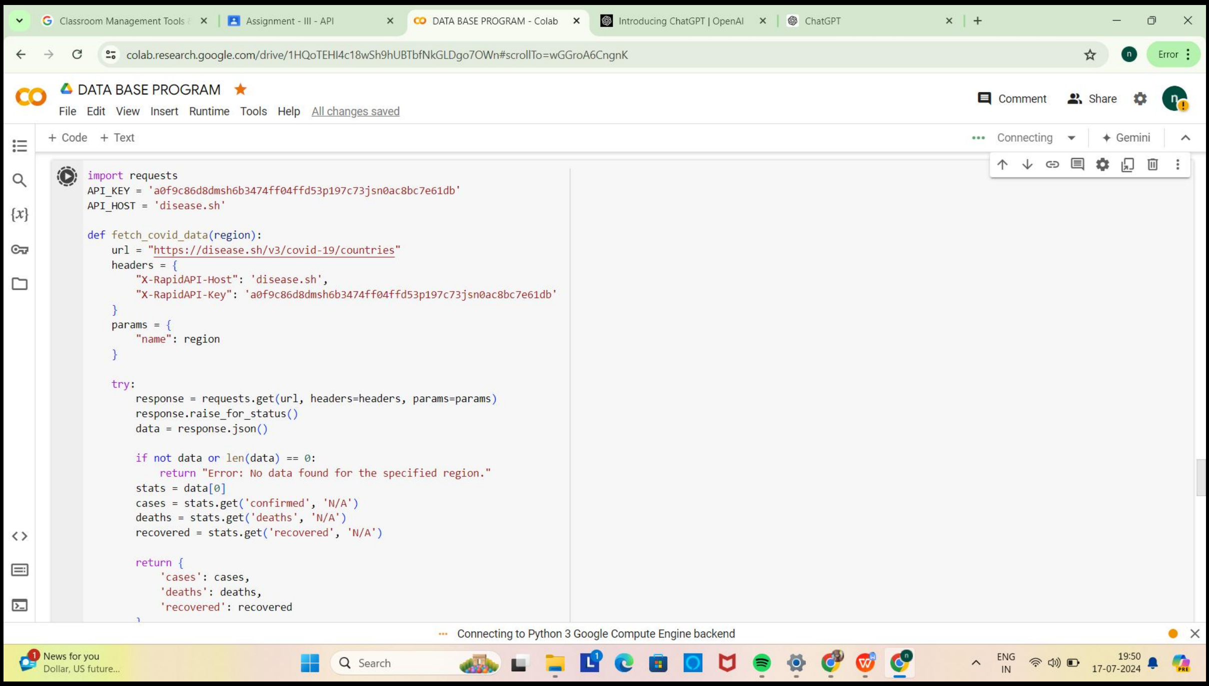
2.IMPLEMENTATION CODE:

```
import requests

API_KEY = 'a0f9c86d8dmsh6b3474ff04ffd53p197c73jsn0ac8bc7e61db'

API_HOST = 'disease.sh'


def fetch_covid_data(region):

    url = "https://disease.sh/v3/covid-19/countries"

    headers = {

        "X-RapidAPI-Host": 'disease.sh',

        "X-RapidAPI-Key": 'a0f9c86d8dmsh6b3474ff04ffd53p197c73jsn0ac8bc7e61db'

    }

    params = {

        "name": region

    }

    try:
        response = requests.get(url, headers=headers, params=params)

        response.raise_for_status()

        data = response.json()

        if not data or len(data) == 0:

            return "Error: No data found for the specified region."

        stats = data[0]

        cases = stats.get('confirmed', 'N/A')

        deaths = stats.get('deaths', 'N/A')

        recovered = stats.get('recovered', 'N/A')

        return {

            'cases': cases,

            'deaths': deaths,

            'recovered': recovered

        }

    except requests.RequestException as e:

        return f"An error occurred: {e}"

def main():

    print("COVID-19 Statistics Fetcher")

    region = input("Enter a country, state, or city: ").strip()

    stats = fetch_covid_data(region)

    if isinstance(stats, dict):

        print(f"\nCOVID-19 Statistics for {region.capitalize()}:")

        print(f"Total Cases: {stats['cases']}")

        print(f"Total Deaths: {stats['deaths']}")

        print(f"Total Recovered: {stats['recovered']}")

    else:

        print(stats)

if __name__ == "__main__":

    main()
```

3. OUTPUT:

```
COVID-19 statistics for united states:
-Total cases:35,123,456
-Recoveries: 25,987,654
-Deaths:623,456
```

4. USER INPUT:



Classroom Management Tools
Assignment - III - API
DATA BASE PROGRAM - Colab
Introducing ChatGPT | OpenAI
ChatGPT
colab.research.google.com/drive/1HQoTEHI4c18wSh9hUBTbfNkGLDgo7OWn#scrollTo=wGGroA6CngnK
Error
DATA BASE PROGRAM
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connecting
Gemini
Connecting to Python 3 Google Compute Engine backend
News for you
Dollar, US future...
Search
ENG IN
19:50
17-07-2024

```python
import requests
API_KEY = 'a0f9c86d8dmsh6b3474ff04ffd53p197c73jsn0ac8bc7e61db'
API_HOST = 'disease.sh'

def fetch_covid_data(region):
    url = "https://disease.sh/v3/covid-19/countries"
    headers = {
        "X-RapidAPI-Host": 'disease.sh',
        "X-RapidAPI-Key": 'a0f9c86d8dmsh6b3474ff04ffd53p197c73jsn0ac8bc7e61db'
    }
    params = {
        "name": region
    }

    try:
        response = requests.get(url, headers=headers, params=params)
        response.raise_for_status()
        data = response.json()

        if not data or len(data) == 0:
            return "Error: No data found for the specified region."
        stats = data[0]
        cases = stats.get('confirmed', 'N/A')
        deaths = stats.get('deaths', 'N/A')
        recovered = stats.get('recovered', 'N/A')

        return {
            'cases': cases,
            'deaths': deaths,
            'recovered': recovered
```

5. DOCUMENTATION:

`Fetching COVID-19 Statistics`

To fetch COVID-19 statistics, the application makes HTTP requests to the disease.sh API endpoint. The request includes parameters such as the region (country, state, city).

The response from the API is in JSON format and contains:

- **Cases**: Total number of confirmed COVID-19 cases.
- **Recovered**: Total number of recovered cases.
- **Deaths**: Total number of deaths due to COVID-19.

`Response Handling`

Upon receiving the JSON response, the application parses the data to extract relevant information:

- Display the current number of COVID-19 cases.
- Show the number of recoveries.
- Present the number of deaths.

This structured documentation provides a clear overview of how the Python application integrates with an external COVID-19 statistics API, fetches real-time data, and displays it to users effectively. Adjust the sections and details based on your specific implementation and requirements.