Assignment: Python Programming

NAME: C PRAVEENA

REGISTER NO: 192324272

DEPARTMENT:

DATE OF SUBMISSION:

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.

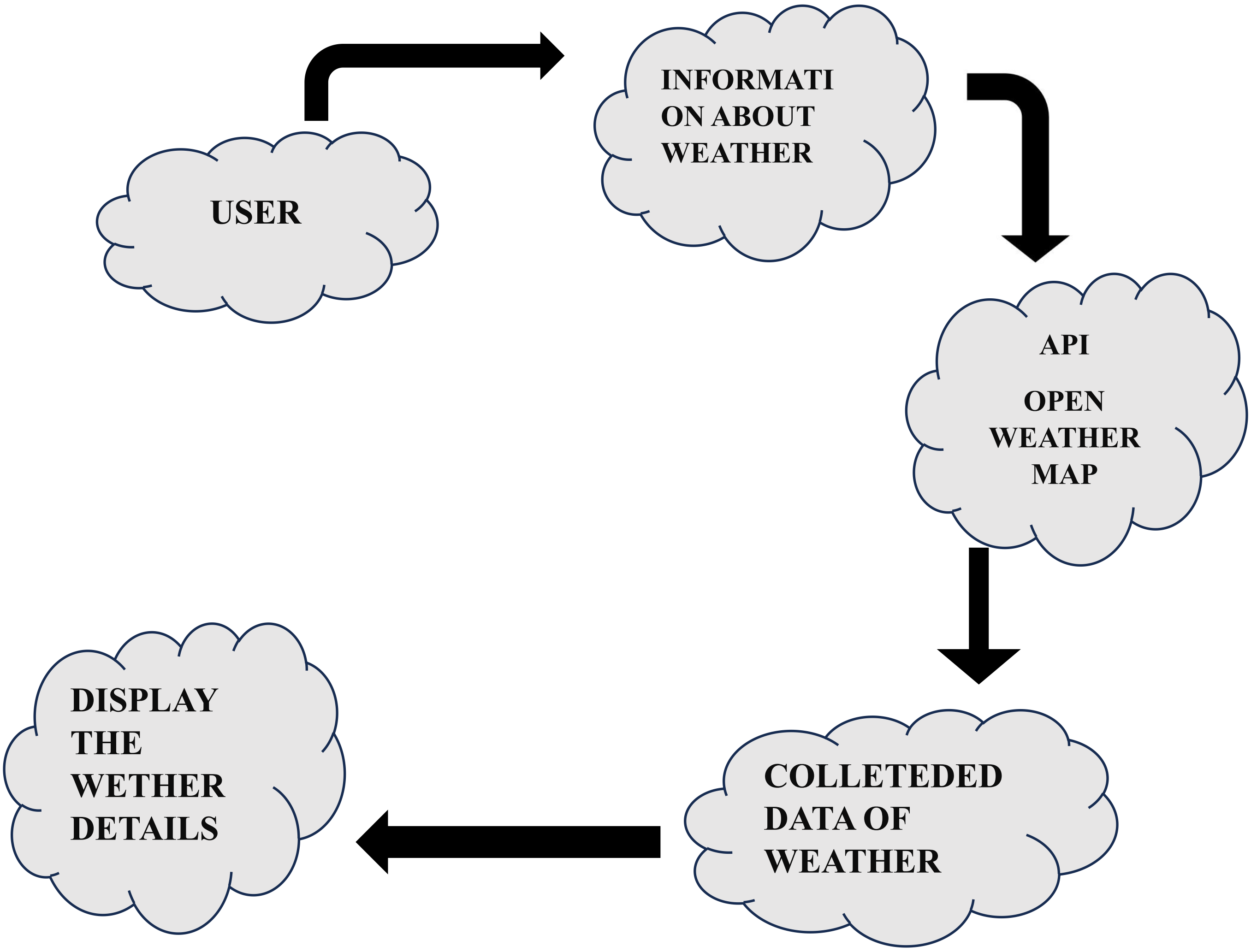
Deliverables:

- 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.

SOLUTIONS:

Real-Time Weather Monitoring System

1.DATA FLOW DIAGRAM:


USER
INFORMATI ON ABOUT WEATHER
API
OPEN WEATHER MAP
COLLETEDED DATA OF WEATHER
DISPLAY THE WETHER DETAILS

2.IMPLIMENTATION:

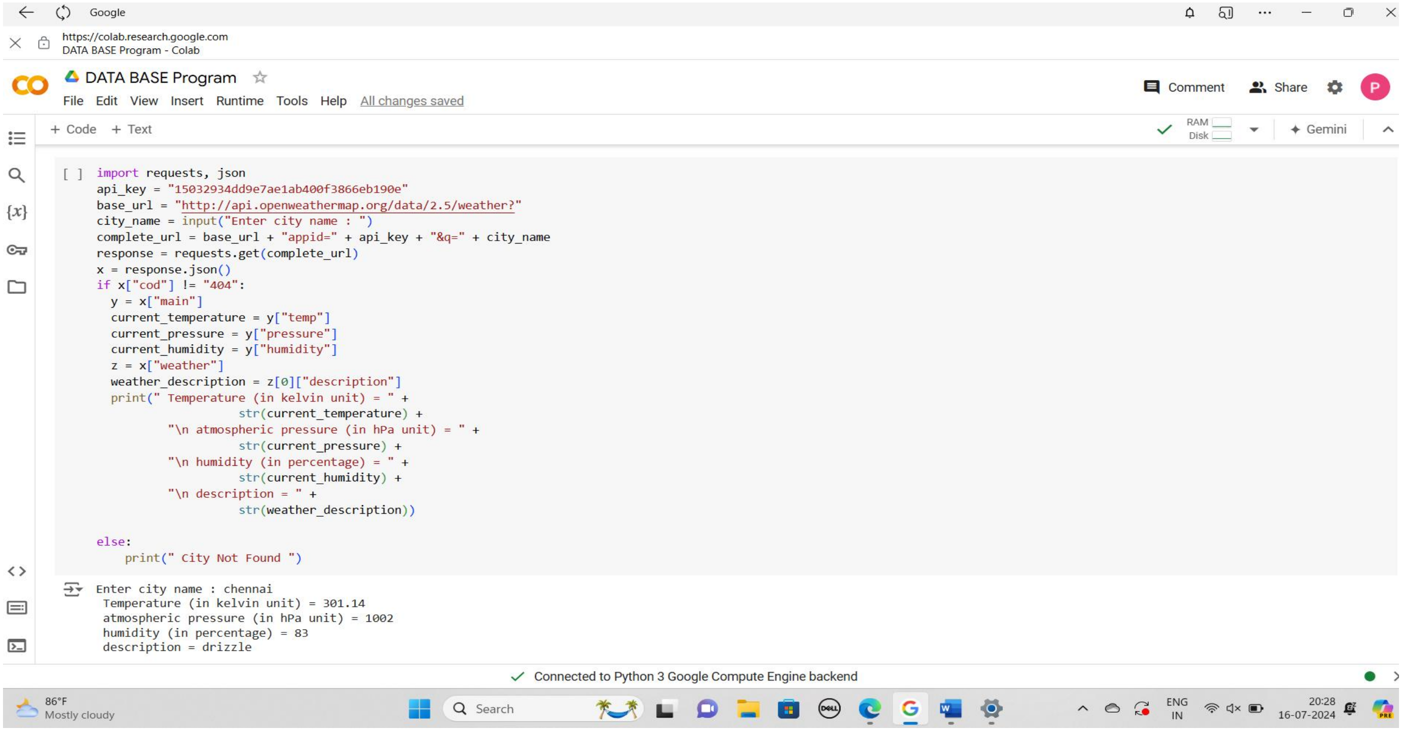
```
import requests, json
api_key = "15032934dd9e7ae1ab400f3866eb190e"
base_url = "http://api.openweathermap.org/data/2.5/weather?"
city_name = input("Enter city name: ")
complete_url = base_url + "appid=" + api_key + "&q=" + city_name
response = requests.get(complete_url)
x = response.json()
if x["cod"] != "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.DISPLAY THE CURRENT WEATHER INFORMATION:

```
Enter city name: chennai
 Temperature (in kelvin unit) = 301.14
 atmospheric pressure (in hPa unit) = 1002
 humidity (in percentage) = 83
 description = drizzle
```

4.USER INPUT:


Google
https://colab.research.google.com
DATA BASE Program - Colab
DATA BASE Program
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Gemini
Connected to Python 3 Google Compute Engine backend
86°F
Mostly cloudy
Search
ENG
IN
20:28
16-07-2024

```
import requests, json
api_key = "15032934dd9e7ae1ab400f3866eb190e"
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 : chennai
 Temperature (in kelvin unit) = 301.14
 atmospheric pressure (in hPa unit) = 1002
 humidity (in percentage) = 83
 description = drizzle
```

5.DOCUMENTATION:

PURPOSE :This document serves as a comprehensive guide for installing, configuring, and using the Real-Time Weather Monitoring System. It is intended for system administrators, developers, and end-users.

SCOPE :The documentation covers all aspects of the system, including hardware setup, software installation, configuration, usage, maintenance, and troubleshooting. Components:

Weather sensors: collect data on temperature, humidity ,wind speed , and other weather pa rameters Data processing unit : process data collected from sensors.

Data base: stores weather data.

DOCUMENT-02

Problem 2: Inventory Management System Optimization

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.

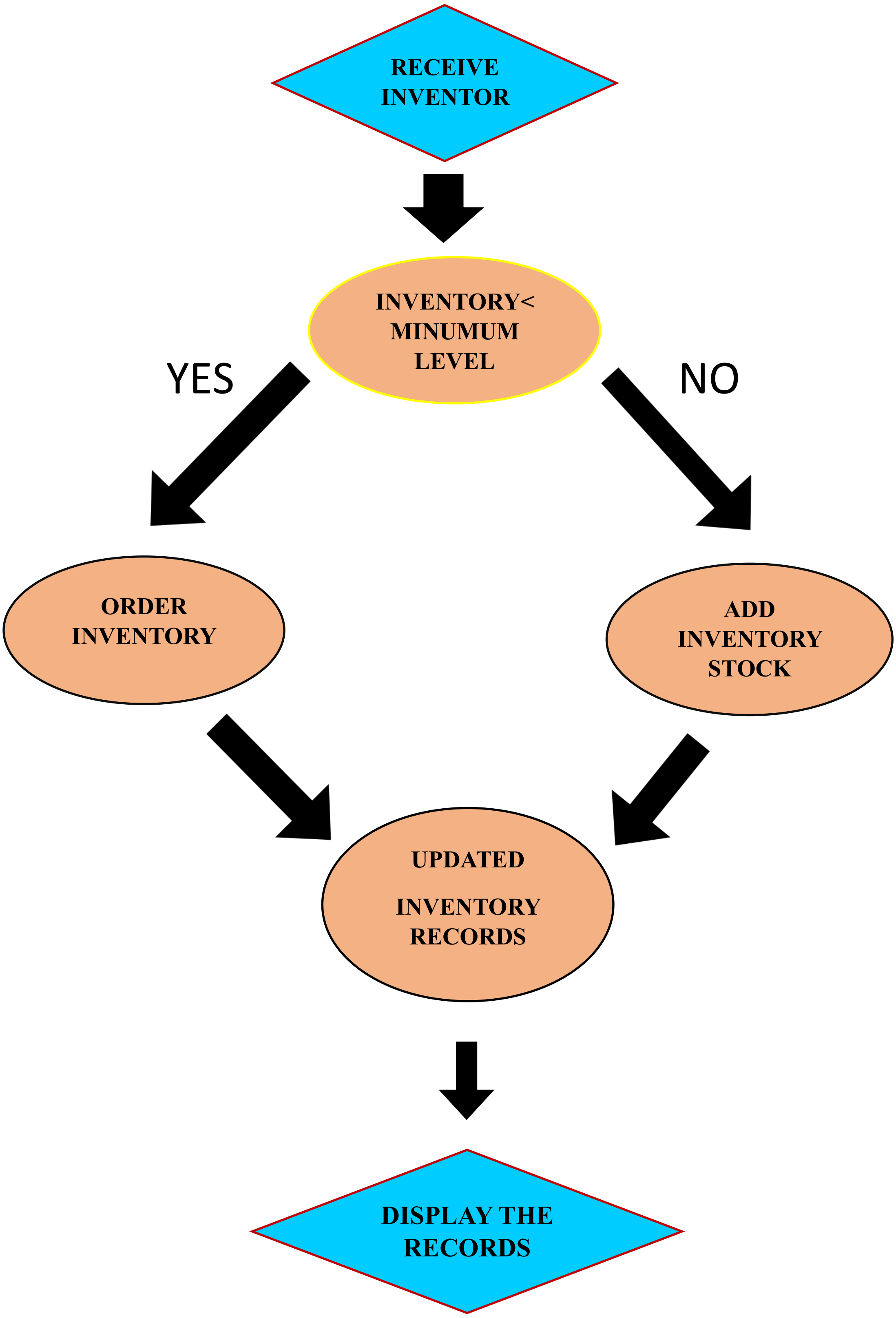
Deliverables:

- **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.

SOLUTION:

Inventory Management System Optimization

1.DATA FLOW DIAGRAM



2.IMPLEMENTATION:

```python
from datetime import datetime
class Product:
    def __init__(self, product_id, name, description, category, price,
supplier):
        self.product_id = product_id
        self.name = name
        self.description = description
        self.category = category
        self.price = price
        self.supplier = supplier
class Warehouse:
    def __init__(self, warehouse_id, name, location):
        self.warehouse_id = warehouse_id
        self.name = name
        self.location = location
class Inventory:
    def __init__(self, product, warehouse, stock_level, reorder_level,
lead_time):
        self.product = product
        self.warehouse = warehouse
        self.stock_level = stock_level
        self.reorder_level = reorder_level
        self.lead_time = lead_time
        self.last_updated = None
products = [
    Product(1, "Laptop", "High-performance laptop", "Electronics", 1200, "Tech
Supplier Inc."),
    Product(2, "Monitor", "27-inch LCD monitor", "Electronics", 300, "Tech
Supplier Inc.")
]
warehouses = [
    Warehouse(1, "Main Warehouse", "New York"),
    Warehouse(2, "Regional Warehouse", "Los Angeles")
]
inventory_items = [
    Inventory(products[0], warehouses[0], 100, 20, 2),
    Inventory(products[1], warehouses[1], 50, 10, 1)
]
def check_stock_levels(product_id):
    for item in inventory_items:
        if item.product.product_id == product_id:
            print(f"Current stock level of {item.product.name}:
{item.stock_level}")
            if item.stock_level < item.reorder_level:
                print(f"Alert: Stock level is below reorder level
({item.reorder_level})")
def update_stock(product_id, warehouse_id, quantity):
    for item in inventory_items:
        if item.product.product_id == product_id and
item.warehouse.warehouse_id == warehouse_id:
            item.stock_level += quantity
            item.last_updated = datetime.now()
            print(f"Stock updated successfully for {item.product.name}. New
stock level: {item.stock_level}")
            return
    print("Product or warehouse not found.")
```

```
def simulate_sales():
    update_stock(1, 1, -5)
    update_stock(2, 2, -2)
def calculate_reorder_quantity(demand_rate, lead_time):
    safety_stock = 10
          reorder_quantity = (demand_rate * lead_time) + safety_stock
    return reorder_quantity
def calculate_all_reorder_quantities():
    for item in inventory_items:
        reorder_quantity = calculate_reorder_quantity(item.stock_level,
item.lead_time)
        print(f"Recommended reorder quantity for {item.product.name}:
{reorder_quantity}")
def generate_inventory_report():
    print("Inventory Report:")
    for item in inventory_items:
        print(f"Product: {item.product.name}, Warehouse:
{item.warehouse.name}, Stock Level: {item.stock_level}")

def generate_overstock_report():
    print("Overstock Report:")
    for item in inventory_items:
        if item.stock_level > item.reorder_level:
            overstock_amount = item.stock_level - item.reorder_level
            print(f"Product: {item.product.name}, Overstock Amount:
{overstock_amount}")
def display_product_info(product_id):
    for product in products:
        if product.product_id == product_id:
            print("Product Information:")
            print(f"Product ID: {product.product_id}")
            print(f"Product Name: {product.name}")
            print(f"Description: {product.description}")
            print(f"Category: {product.category}")
            print(f"Price: {product.price}")
            print(f"Supplier: {product.supplier}")
            return

def main():
    print("Welcome to Inventory Tracking System!")
    while True:
        print("\nMenu:")
        print("1. Check Stock Levels")
        print("2. Update Stock Levels")
        print("3. Simulate Sales/Usage")
        print("4. Calculate Reorder Quantities")
        print("5. Generate Reports")
        print("6. View Product Information")
        print("7. Exit")

        choice = input("Enter your choice (1-7): ")
```

```
 if choice == '1':
            product_id = int(input("Enter product ID to check stock levels:
"))
            check_stock_levels(product_id)
        elif choice == '2':
            product_id = int(input("Enter product ID to update stock: "))
            warehouse_id = int(input("Enter warehouse ID: "))
            quantity = int(input("Enter quantity to add/subtract (use
negative for subtracting): "))
            update_stock(product_id, warehouse_id, quantity)
        elif choice == '3':
            simulate_sales()
        elif choice == '4':
            calculate_all_reorder_quantities()
        elif choice == '5':
            report_choice = input("Choose report type (1 - Inventory Report,
2 - Overstock Report): ")
            if report_choice == '1':
                generate_inventory_report()
            elif report_choice == '2':
                generate_overstock_report()
            else:
                print("Invalid choice.")
        elif choice == '6':
            product_id = int(input("Enter product ID to view information:
"))
            display_product_info(product_id)
        elif choice == '7':
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please enter a valid option.")
if __name__ == "__main__":
    main()
```

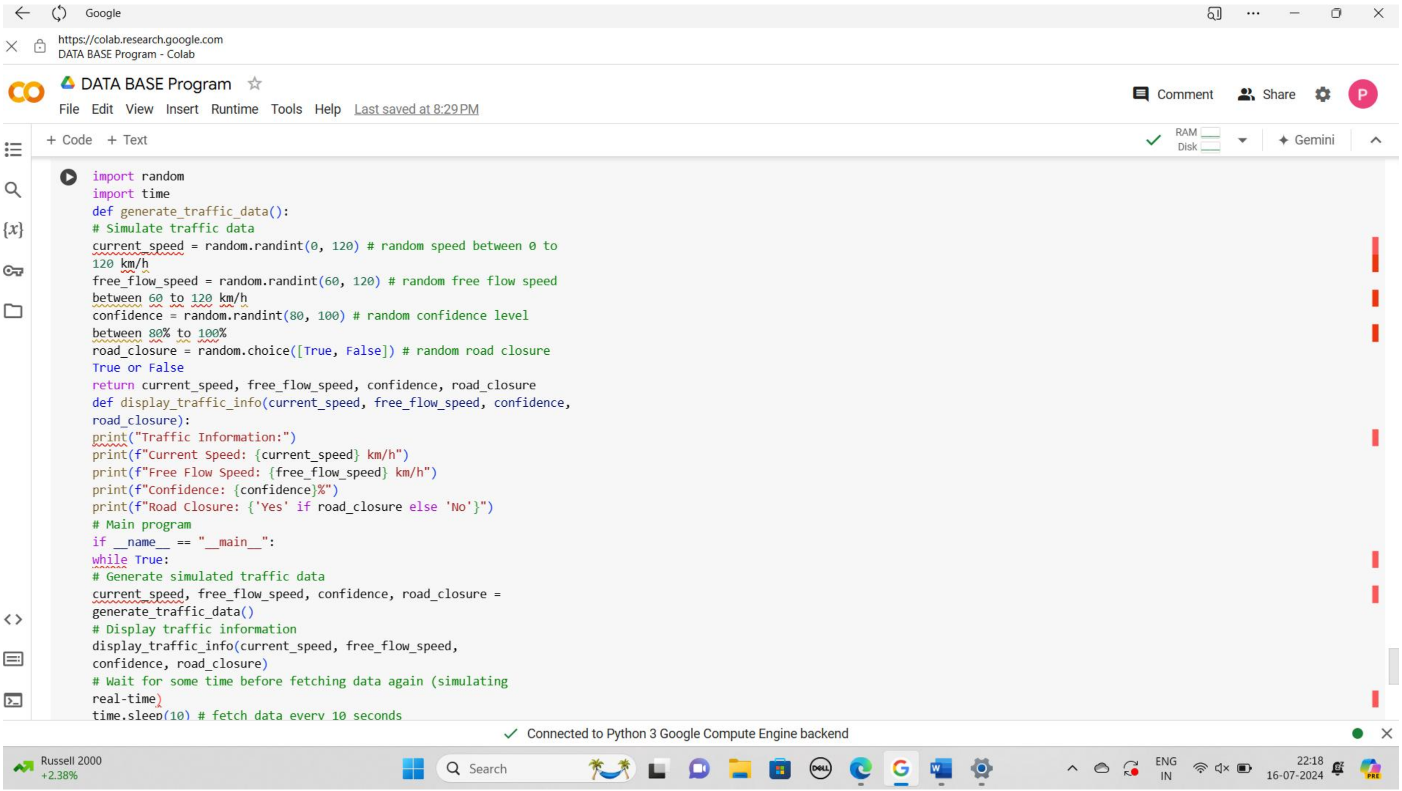
3.DISPLAY THE OUTPUT:

```
Welcome to Inventory Tracking System!

Menu:
1. Check Stock Levels
2. Update Stock Levels
3. Simulate Sales/Usage
4. Calculate Reorder Quantities
5. Generate Reports
6. View Product Information
7. Exit
Recommended reorder quantity for Laptop: 210
Recommended reorder quantity for Monitor: 60

Menu:
1. Check Stock Levels
2. Update Stock Levels
3. Simulate Sales/Usage
4. Calculate Reorder Quantities
5. Generate Reports
6. View Product Information
7. Exit
```

4.USER INPUT:


Google
https://colab.research.google.com
DATA BASE Program - Colab
DATA BASE Program
File Edit View Insert Runtime Tools Help Last saved at 8:29 PM
Comment Share
+ Code + Text
RAM
Disk
Gemini

```
import random
import time
def generate_traffic_data():
# Simulate traffic data
current_speed = random.randint(0, 120) # random speed between 0 to
120 km/h
free_flow_speed = random.randint(60, 120) # random free flow speed
between 60 to 120 km/h
confidence = random.randint(80, 100) # random confidence level
between 80% to 100%
road_closure = random.choice([True, False]) # random road closure
True or False
return current_speed, free_flow_speed, confidence, road_closure
def display_traffic_info(current_speed, free_flow_speed, confidence,
road_closure):
print("Traffic Information:")
print(f"Current Speed: {current_speed} km/h")
print(f"Free Flow Speed: {free_flow_speed} km/h")
print(f"Confidence: {confidence}%")
print(f"Road Closure: {'Yes' if road_closure else 'No'}")
# Main program
if __name__ == "__main__":
while True:
# Generate simulated traffic data
current_speed, free_flow_speed, confidence, road_closure =
generate_traffic_data()
# Display traffic information
display_traffic_info(current_speed, free_flow_speed,
confidence, road_closure)
# Wait for some time before fetching data again (simulating
real-time)
time.sleep(10) # fetch data every 10 seconds
```

Connected to Python 3 Google Compute Engine backend

5.DOCUMENTATION:

Boost Accuracy: Make sure inventory records are current and correct. Cut Expenses: Keep holding and ordering expenses to a minimum.

Boost Efficiency: To save time and effort, simplify inventory operations. Boost client satisfaction by making sure products are available to satisfy needs from customers.

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.

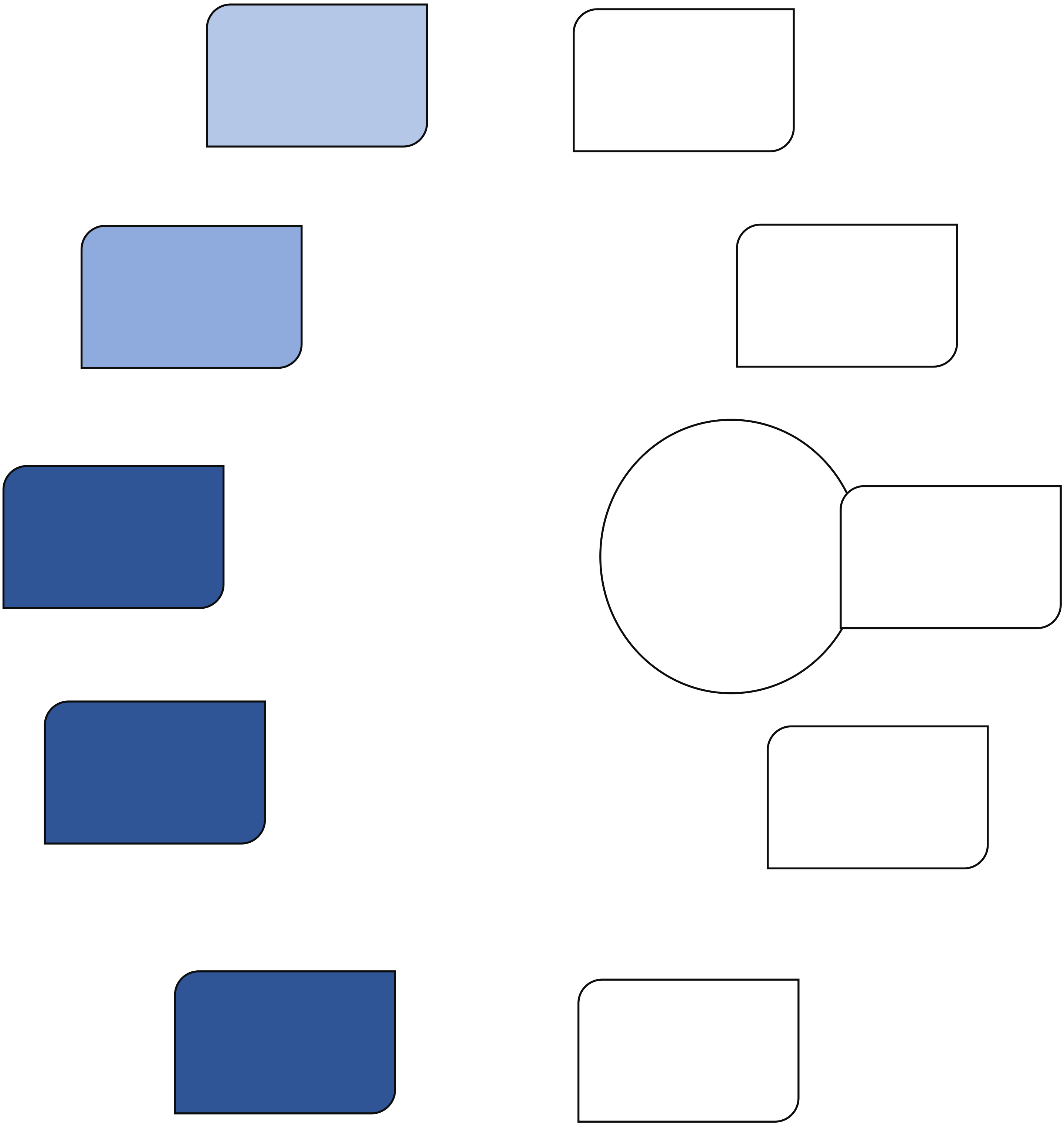
Deliverables:

- 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**.**

SOLUTIONS:

Real-Time Traffic Monitoring System

1.DATA FLOW DIAGRAM



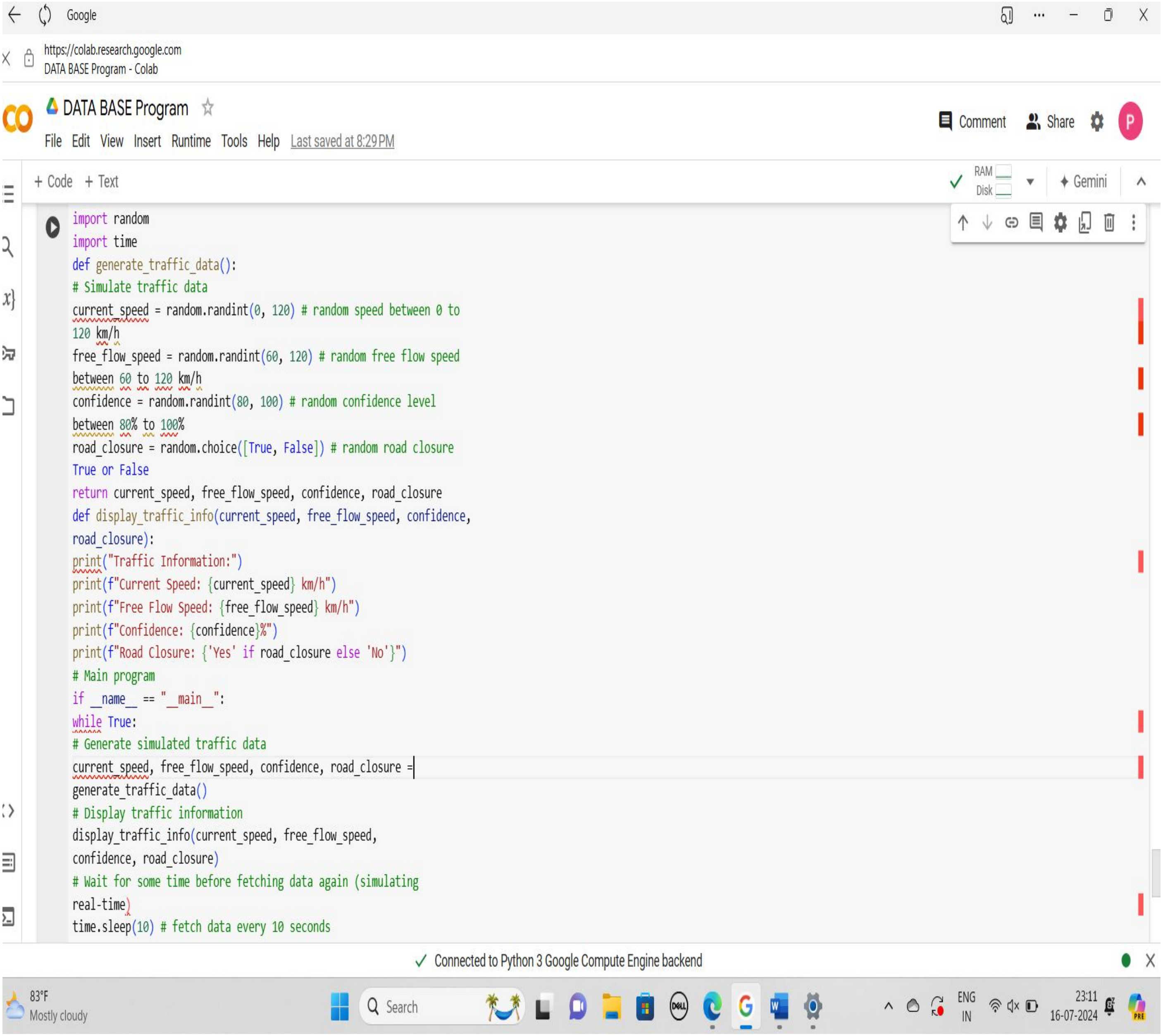
2.IMPLEMENTATION

```
import random
import time
def generate_traffic_data():
# Simulate traffic data
current_speed = random.randint(0, 120) # random speed between 0 to
120 km/h
free_flow_speed = random.randint(60, 120) # random free flow speed
between 60 to 120 km/h
confidence = random.randint(80, 100) # random confidence level
between 80% to 100%
road_closure = random.choice([True, False]) # random road closure
True or False
return current_speed, free_flow_speed, confidence, road_closure
def display_traffic_info(current_speed, free_flow_speed, confidence,
road_closure):
print("Traffic Information:")
print(f"Current Speed: {current_speed} km/h")
print(f"Free Flow Speed: {free_flow_speed} km/h")
print(f"Confidence: {confidence}%")
print(f"Road Closure: {'Yes' if road_closure else 'No'}")
# Main program
if __name__ == "__main__":
while True:
# Generate simulated traffic data
current_speed, free_flow_speed, confidence, road_closure =
generate_traffic_data()
# Display traffic information
display_traffic_info(current_speed, free_flow_speed,
confidence, road_closure)
# Wait for some time before fetching data again (simulating
real-time)
time.sleep(10) # fetch data every 10 seconds
```

3.DISPLAY THE OUTPUT:

```
Traffic Information: Current Speed: 113 km/h Free Flow Speed: 120 km/h Confidence: 94% Road Closure: No
Traffic Information: Current Speed: 117 km/h Free Flow Speed: 79 km/h Confidence: 90% Road Closure: No
Traffic Information: Current Speed: 58 km/h Free Flow Speed: 109 km/h Confidence: 94% Road Closure: Yes
Traffic Information: Current Speed: 113 km/h Free Flow Speed: 120 km/h Confidence: 94% Road Closure: No
```

4.USER INPUT:



```
import random
import time
def generate_traffic_data():
# Simulate traffic data
current_speed = random.randint(0, 120) # random speed between 0 to
120 km/h
free_flow_speed = random.randint(60, 120) # random free flow speed
between 60 to 120 km/h
confidence = random.randint(80, 100) # random confidence level
between 80% to 100%
road_closure = random.choice([True, False]) # random road closure
True or False
return current_speed, free_flow_speed, confidence, road_closure
def display_traffic_info(current_speed, free_flow_speed, confidence,
road_closure):
print("Traffic Information:")
print(f"Current Speed: {current_speed} km/h")
print(f"Free Flow Speed: {free_flow_speed} km/h")
print(f"Confidence: {confidence}%")
print(f"Road Closure: {'Yes' if road_closure else 'No'}")
# Main program
if __name__ == "__main__":
while True:
# Generate simulated traffic data
current_speed, free_flow_speed, confidence, road_closure =
generate_traffic_data()
# Display traffic information
display_traffic_info(current_speed, free_flow_speed,
confidence, road_closure)
# Wait for some time before fetching data again (simulating
real-time)
time.sleep(10) # fetch data every 10 seconds
```

✓ Connected to Python 3 Google Compute Engine backend

5.DOCUMENTATION:

Objectives : Give Users Accurate Traffic Data: Give users access to real-time traffic data for efficient planning and navigation. Boost Road Safety: Increase user awareness of traffic accidents, road closures, and dangerous situations to improve road safety. Optimize Traffic Flow: With data-driven insights and suggestions, help manage traffic flow and lessen congestion. Highlights Dashboard Synopsis Real-time traffic data: Use color-coded maps to show the speed of traffic, the amount of congestion, and incidents.

Traffic projections: Using historical data and in-the-moment analytics, provide both short and long-term traffic projections.

DOCUMENT-04

Problem 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.

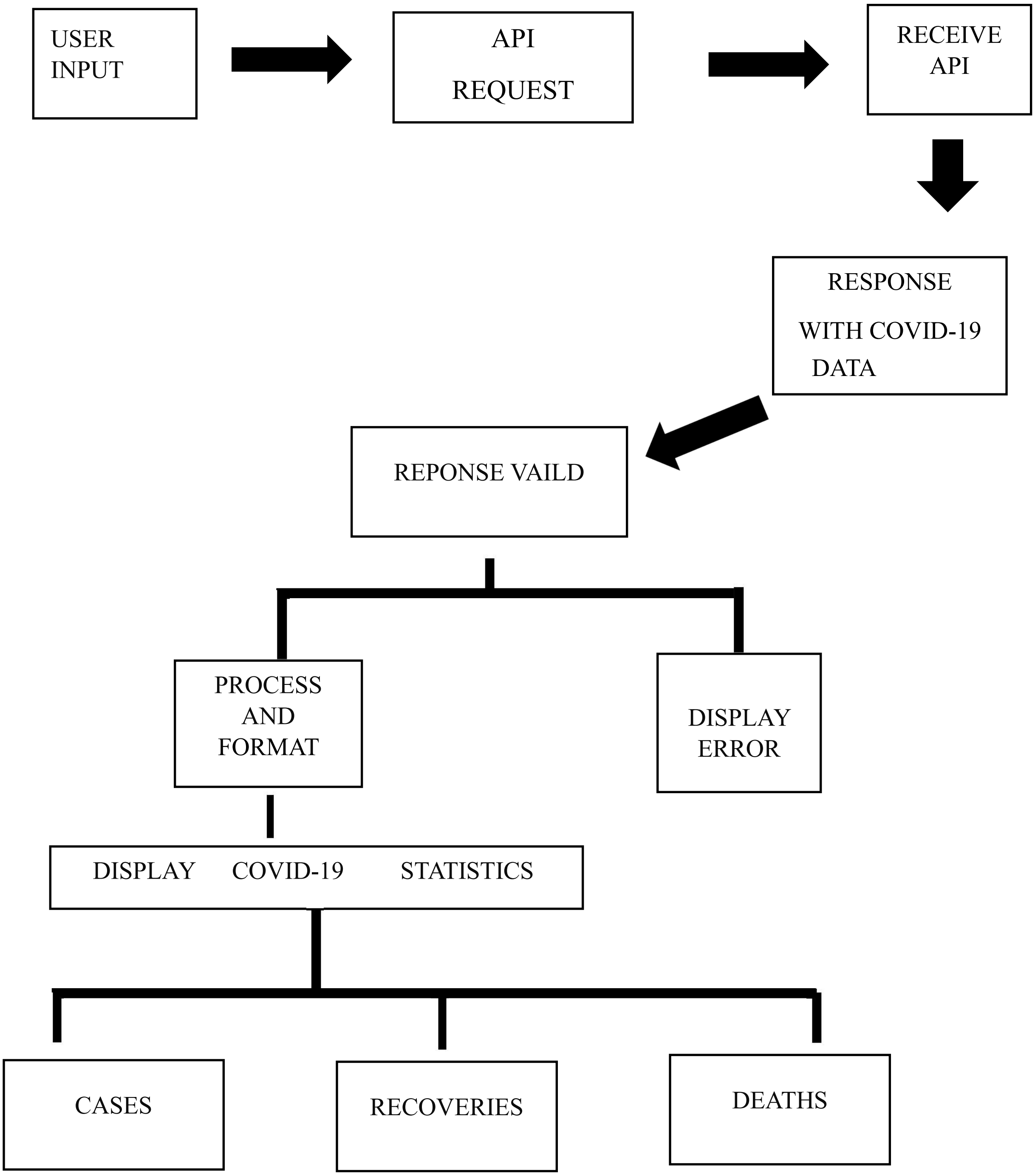
Deliverables:

- 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 improvements.

SOLUTION:

Real-Time COVID-19 Statistics Tracker

1.DATA FLOW DIAGRAM:



2.IMPLEMENTATION:

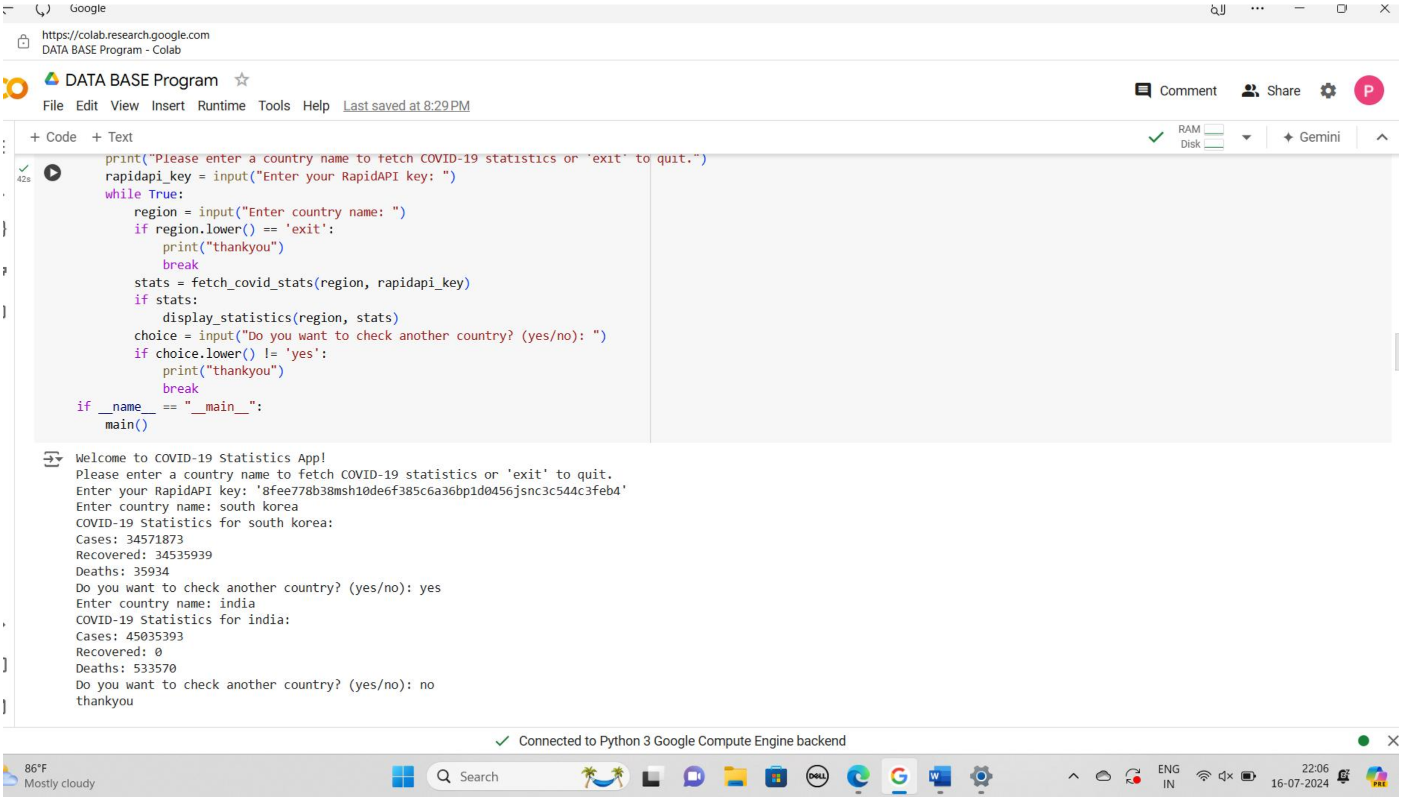
```
import requests
def fetch_covid_stats(region, rapidapi_key):
    base_url = "https://disease.sh/v3/covid-19"
    endpoint = "/countries/" + region
    url = base_url + endpoint
    headers = {
```

```
    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            data = response.json()
            cases = data['cases']
            recovered = data['recovered']
            deaths = data['deaths']
            return (cases, recovered, deaths)
        else:
            print(f"Error fetching data for {region}. Status code: 
{response.status_code}")
            return None
    except requests.exceptions.RequestException as e:
        print(f"Error fetching data: {str(e)}")
        return None
def display_statistics(region, stats):
    cases, recovered, deaths = stats
    print(f"COVID-19 Statistics for {region}:")
    print(f"Cases: {cases}")
    print(f"Recovered: {recovered}")
    print(f"Deaths: {deaths}")
def main():
    print("Welcome to COVID-19 Statistics App!")
    print("Please enter a country name to fetch COVID-19 statistics or 'exit' 
to quit.")
    rapidapi_key = input("Enter your RapidAPI key: ")
    while True:
        region = input("Enter country name: ")
        if region.lower() == 'exit':
            print("thankyou")
            break
        stats = fetch_covid_stats(region, rapidapi_key)
        if stats:
            display_statistics(region, stats)
        choice = input("Do you want to check another country? (yes/no): ")
        if choice.lower() != 'yes':
            print("thankyou")
            break
if __name__ == "__main__":
    main()
```

3.DISPLAY THE OUTPUT:

```
Welcome to COVID-19 Statistics App!
Please enter a country name to fetch COVID-19 statistics or 'exit' to quit.
Enter your RapidAPI key: '8fee778b38msh10de6f385c6a36bp1d0456jsnc3c544c3feb4
Enter country name: south korea
COVID-19 Statistics for south korea:
Cases: 34571873
Recovered: 34535939
Deaths: 35934
Do you want to check another country? (yes/no): yes
Enter country name: india
Do you want to check another country? (yes/no): no
COVID-19 Statistics for india:
Cases: 45035393
```

4.USER INPUT:


Google
https://colab.research.google.com
DATA BASE Program - Colab
DATA BASE Program
File Edit View Insert Runtime Tools Help Last saved at 8:29 PM
Comment
Share
+ Code + Text
RAM
Disk
Gemini
42s
Connected to Python 3 Google Compute Engine backend
86°F
Mostly cloudy
Search
ENG
IN
22:06
16-07-2024

```
    print("Please enter a country name to fetch COVID-19 statistics or 'exit' to quit.")
    rapidapi_key = input("Enter your RapidAPI key: ")
    while True:
        region = input("Enter country name: ")
        if region.lower() == 'exit':
            print("thankyou")
            break
        stats = fetch_covid_stats(region, rapidapi_key)
        if stats:
            display_statistics(region, stats)
        choice = input("Do you want to check another country? (yes/no): ")
        if choice.lower() != 'yes':
            print("thankyou")
            break
if __name__ == "__main__":
    main()
```

```
Welcome to COVID-19 Statistics App!
Please enter a country name to fetch COVID-19 statistics or 'exit' to quit.
Enter your RapidAPI key: '8fee778b38msh10de6f385c6a36bp1d0456jsnc3c544c3feb4'
Enter country name: south korea
COVID-19 Statistics for south korea:
Cases: 34571873
Recovered: 34535939
Deaths: 35934
Do you want to check another country? (yes/no): yes
Enter country name: india
COVID-19 Statistics for india:
Cases: 45035393
Recovered: 0
Deaths: 533570
Do you want to check another country? (yes/no): no
thankyou
```

5.DOCUMENTATION:

OBJECTIVES: By giving users precise information, we hope to guarantee that they have quick and accurate access to COVID-19 data from reliable sources.

Increase Public Awareness: Inform people about the patterns and effects of COVID-19.

Facilitate Decision-Making: By offering current data, you may help the general public, healthcare professionals, and lawmakers make well-informed decisions. Highlights dashboard Synopsis Global Statistics: Show the total number of confirmed cases, deaths, recoveries, and ongoing cases globally. Regional Breakdown: Include interactive maps and charts with statistics for particular regions or nations. Analyze trends over time with graphs that display COVID-19 metrics changes on a daily, weekly, and monthly