Assignment:3 Python Programming

NAME: P SRAVANTHI

REGISTER NUMBER: 192311272

DEPARTMENT: CSE

DATE OF SUBMISSION: 17/07/2024

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., Open Weather Map) 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.

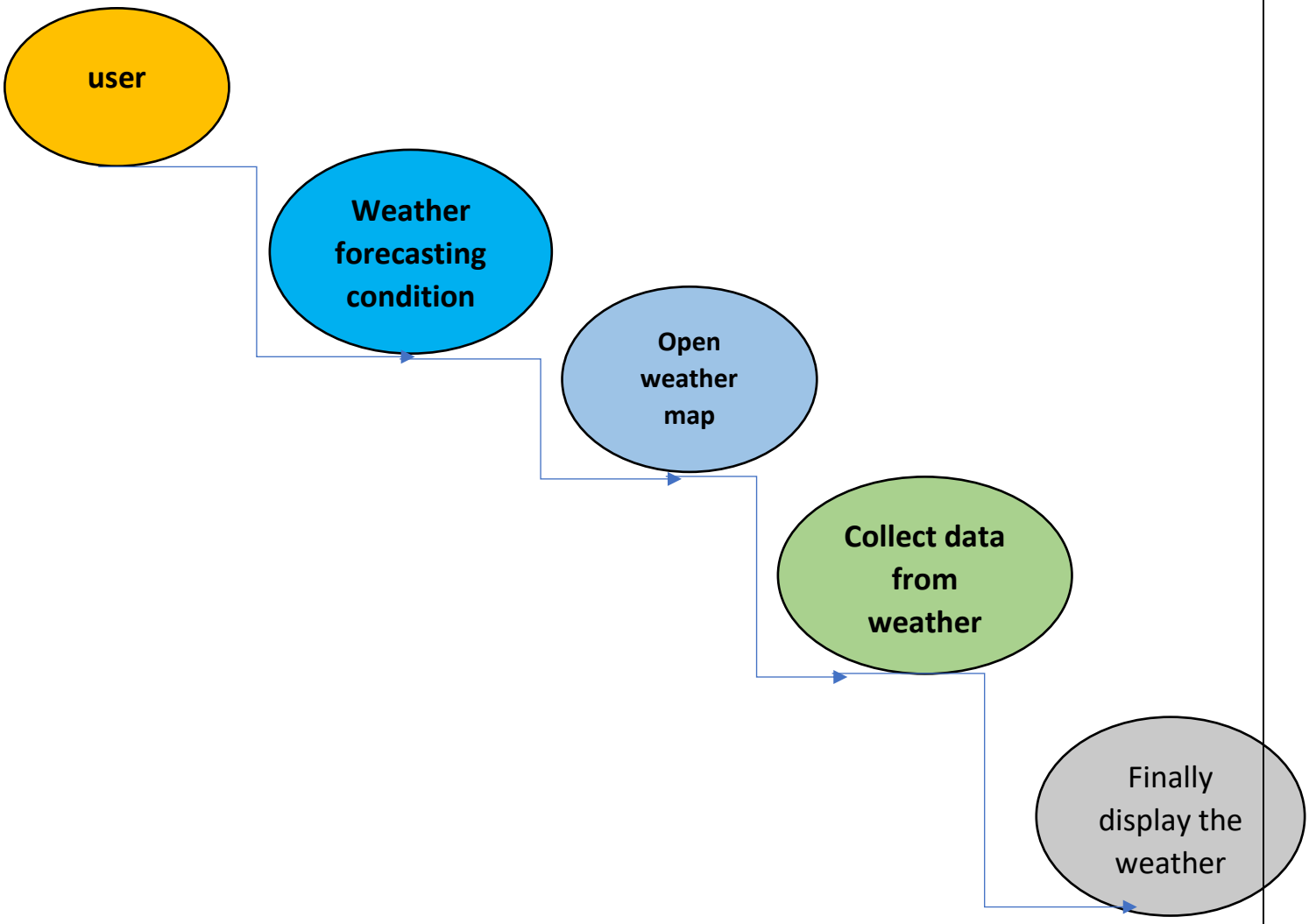
• Documentation of the API integration and the methods used to fetch and display weather data.

• Explanation of any assumptions made and potential improvements.

Solution:

Real-Time Weather Monitoring System

1.Data Flow Diagram



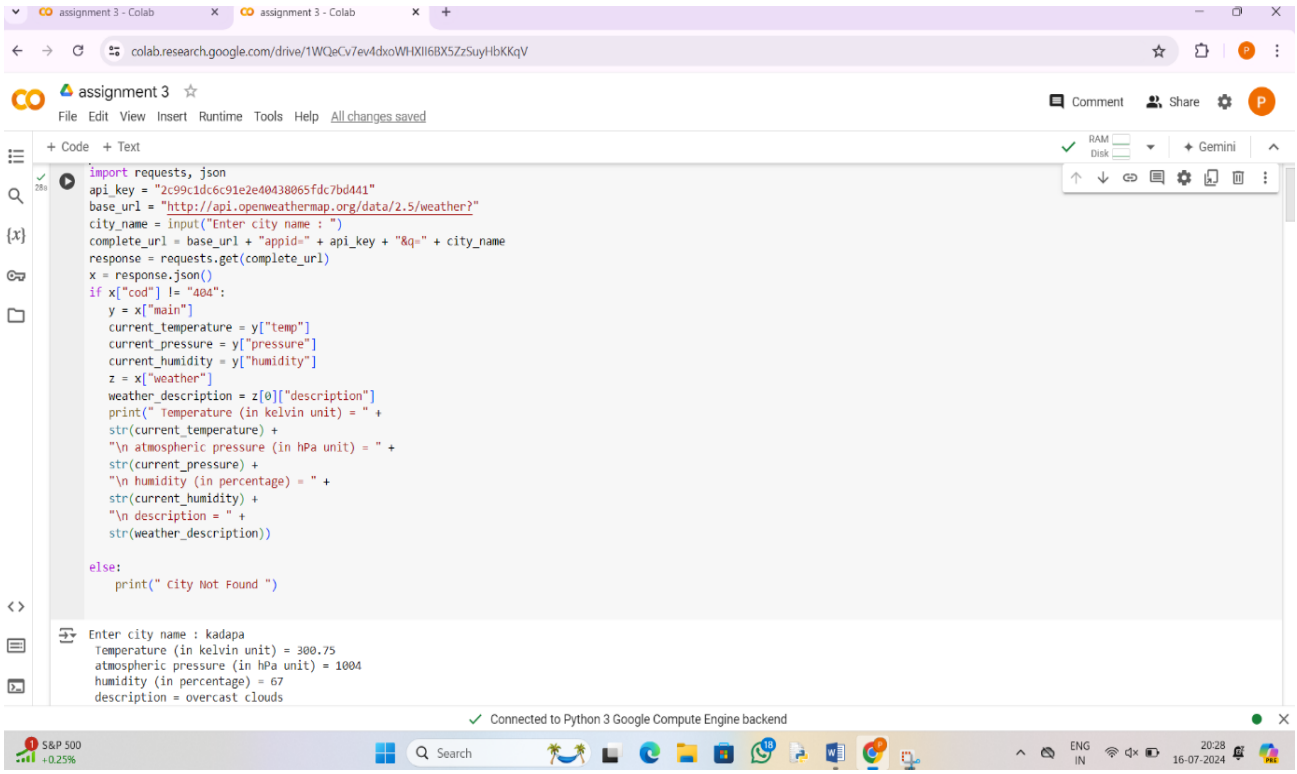
2.Implementation

```
import requests, json
api_key = "2c99c1dc6c91e2e40438065fdc7bd441"
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 : Kadapa.
Temperature (in kelvin unit) = 300.75
atmospheric pressure (in hPa unit) = 1004
humidity (in percentage) = 67
description = overcast clouds

4.User Input



5.DOCUMENTATION:

Purpose: The purpose of a Real-Time Weather Monitoring System is to collect, analyse , and disseminate weather data to provide timely and accurate weather information. This system supports various applications, including agriculture, disaster management, urban planning, and daily life decision-making.

Scope: The scope of the system includes:

Data Collection: Gathering weather data from multiple sources (sensors, satellites, weather stations).

Data Storage: Maintaining a reliable database for historical and real-time weather data. Components :

Weather Sensor: types include temperature, humidity, pressure, wind direction and speed, rainfall, and sun radiation.
Function: Gather environmental data in real time.
Data base: store weather data

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).

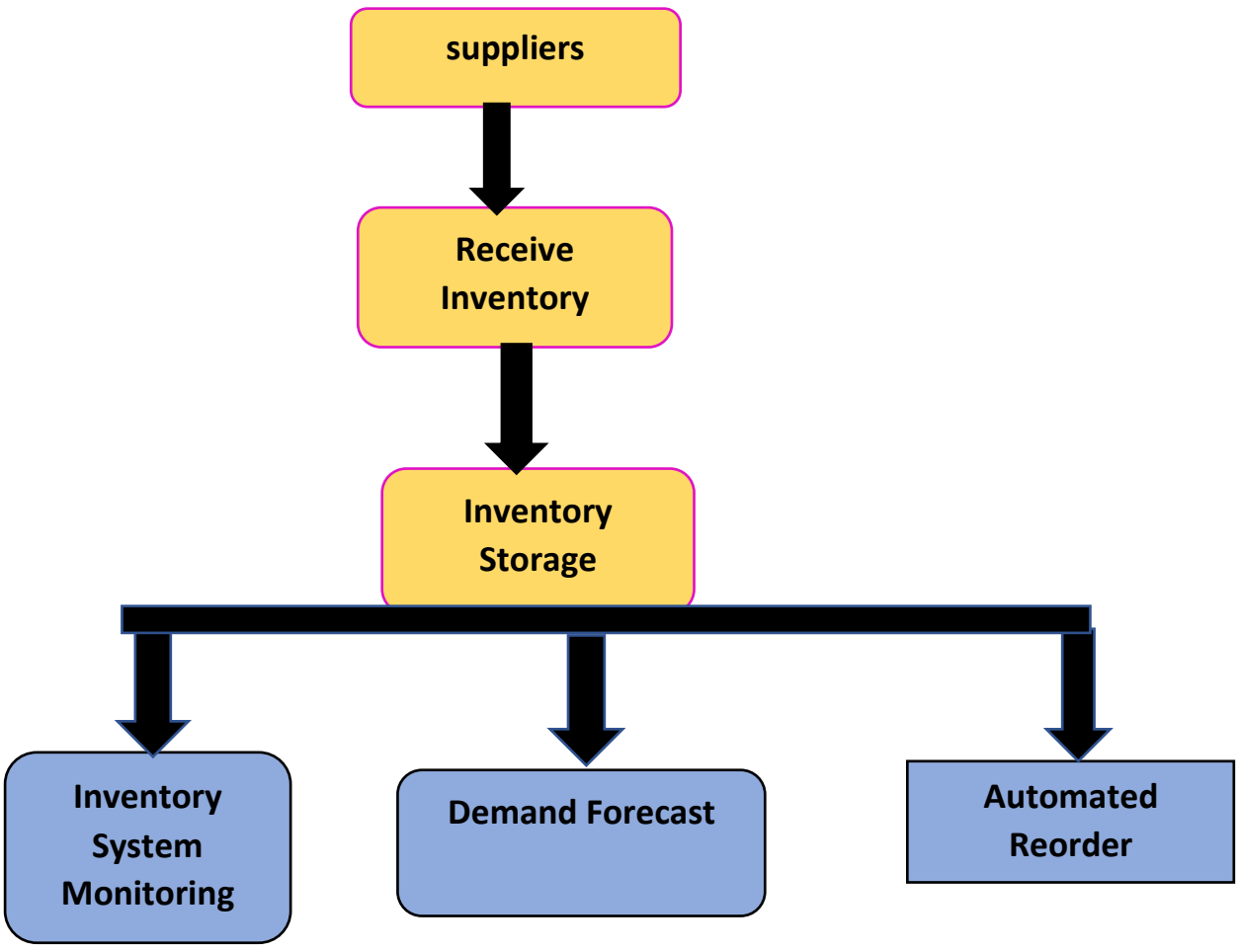
• User Interface: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.

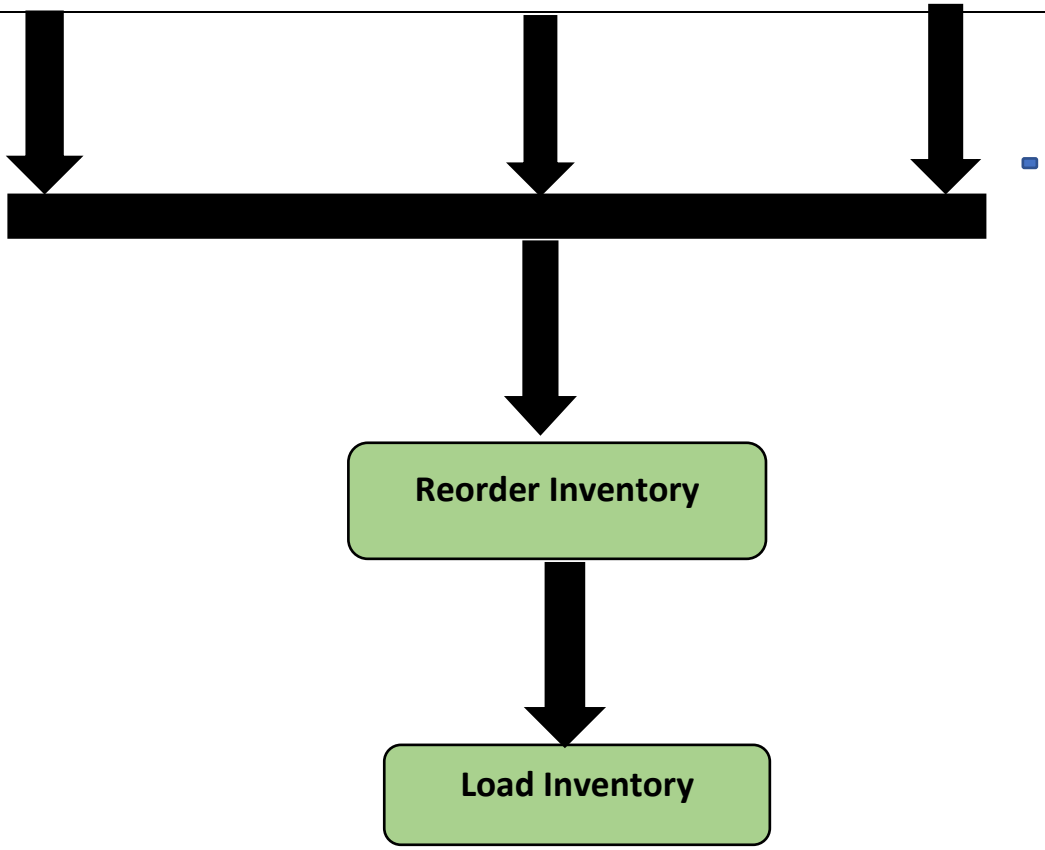
• Assumptions and Improvements: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

Solution:

Inventory Management System Optimization

1.Data Flow Diagram





2.Implementation

```
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 inventory management system optimization

Welcome to Inventory Tracking System!

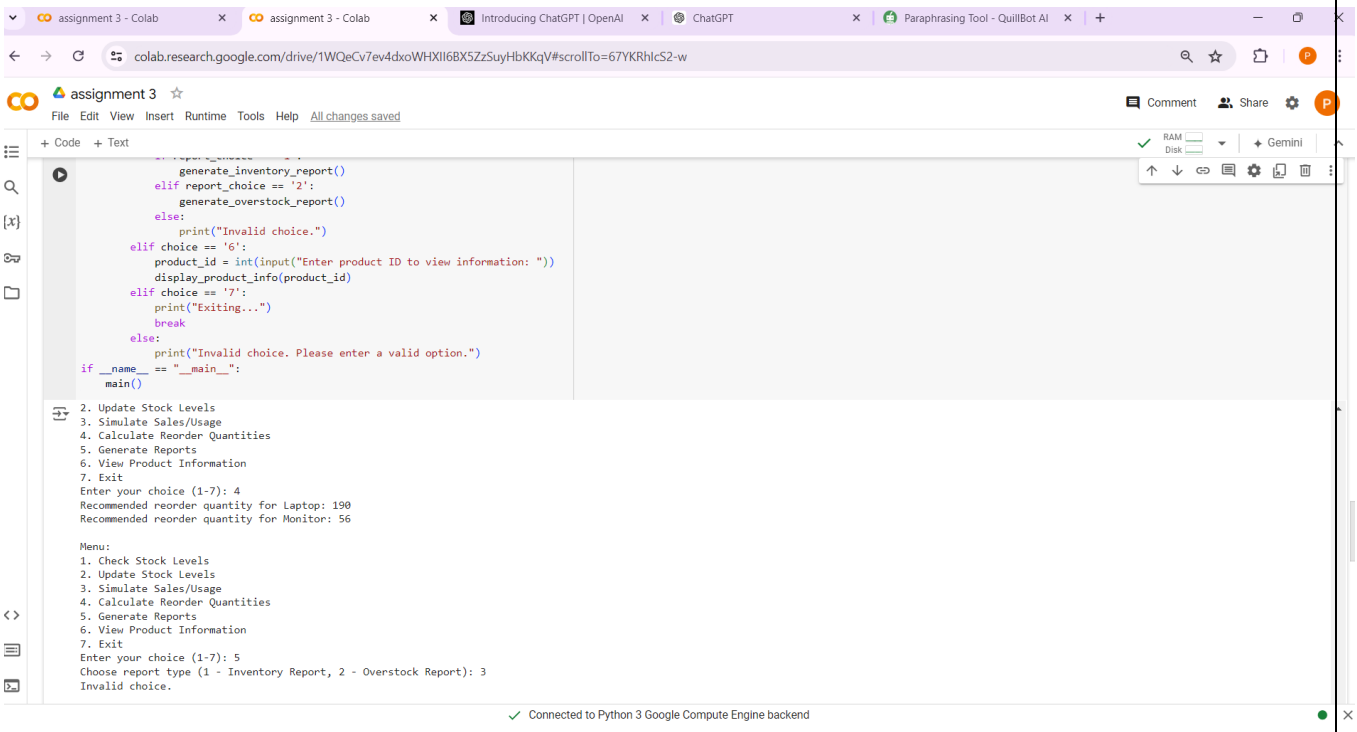
Menu:
1. Check Stock Levels
2. Update Stock Levels
3. Simulate Sales/Usage
4. Calculate Reorder Quantities
5. Generate Reports
6. View Product Information
7. Exit
Enter your choice (1-7): 4
Recommended reorder quantity for Laptop: 210
Recommended reorder quantity for Monitor: 60

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
Enter your choice (1-7): 3
Stock updated successfully for Laptop. New stock level: 95
Stock updated successfully for Monitor. New stock level: 48

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
Enter your choice (1-7): 2
Enter product ID to update stock: 4
Enter warehouse ID: 5

Enter quantity to add/subtract (use negative for subtracting): 7
Product or warehouse not found.

4.User Input



5.Documentation

Enhance Accuracy: Verify that inventory data are accurate and up to date.
Decrease Charges: Keep ordering and holding prices as low as possible.
Boost Efficiency: Simplify inventory processes to save time and effort.

Improve customer satisfaction by making sure products are available to meet demands from clients.

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.

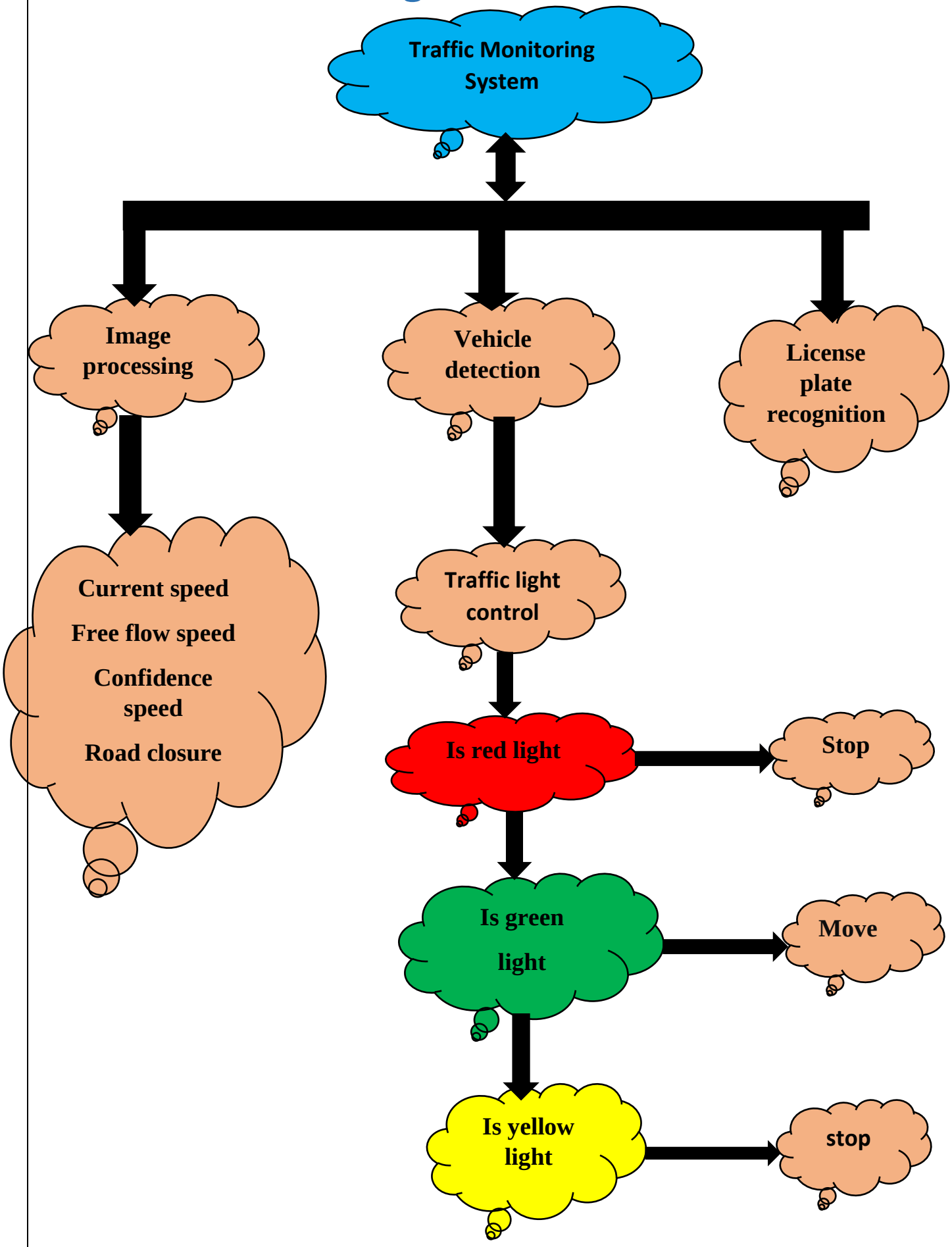
• Documentation of the API integration and the methods used to fetch and display traffic data.

• Explanation of any assumptions made and potential improvements.

Solution:

Real-Time Traffic Monitoring System

1.Data Flow Diagram



2.Implementation

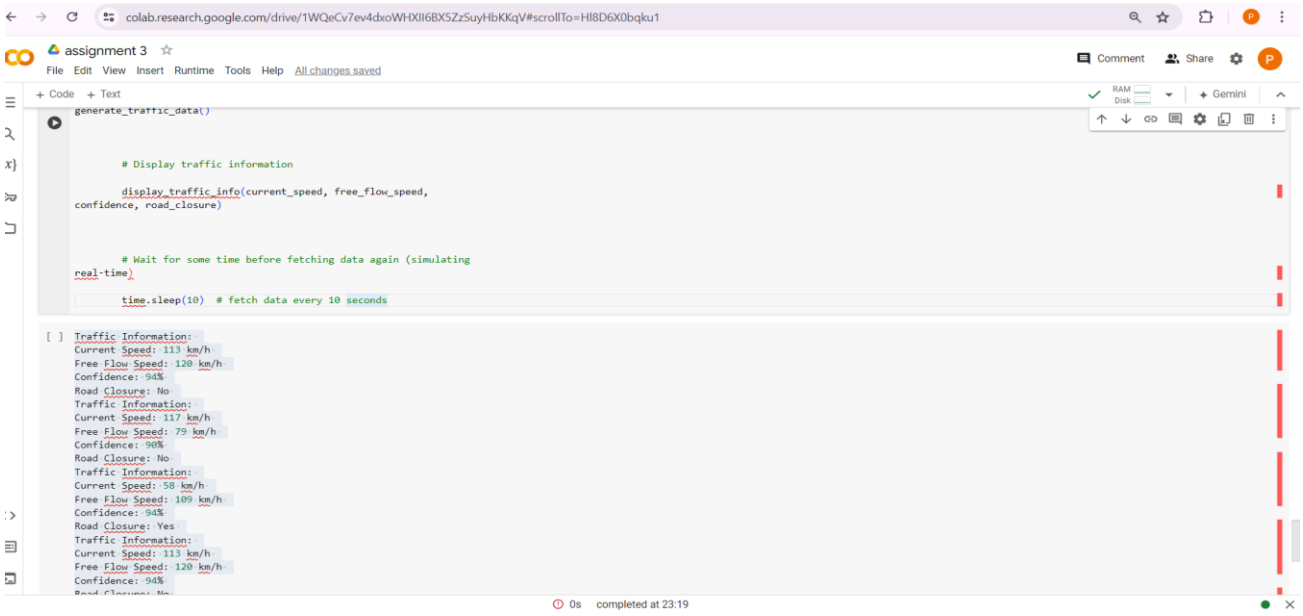
```
import random

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 Real-Time Traffic Monitoring System

```
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: 118 km/h
Free Flow Speed: 125 km/h
Confidence: 94%
Road Closure: No
```

4.User Input



5.Documentation:

objectives: Provide Users with Accurate Traffic Information: Provide users with access to up-to-date traffic information for effective planning and navigating.
Improved Traffic Safety Inform users about road closures, traffic accidents, and hazardous

situations to raise the level of traffic safety.
Optimise Traffic Flow: Manage traffic flow and reduce congestion using data-driven insights and suggestions.
Highlights Overview of the Dashboard
Data about traffic in real time: Maps using colour codes can be used to display traffic volume, incident frequency, and speed.
Traffic forecasts: Provide both short- and long-term traffic projections based on historical data and real-time analytics.

Program 4: Real-Time COVID-19 Statistics Tracker

Scenario:

You are developing a real-timeCOVID-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.

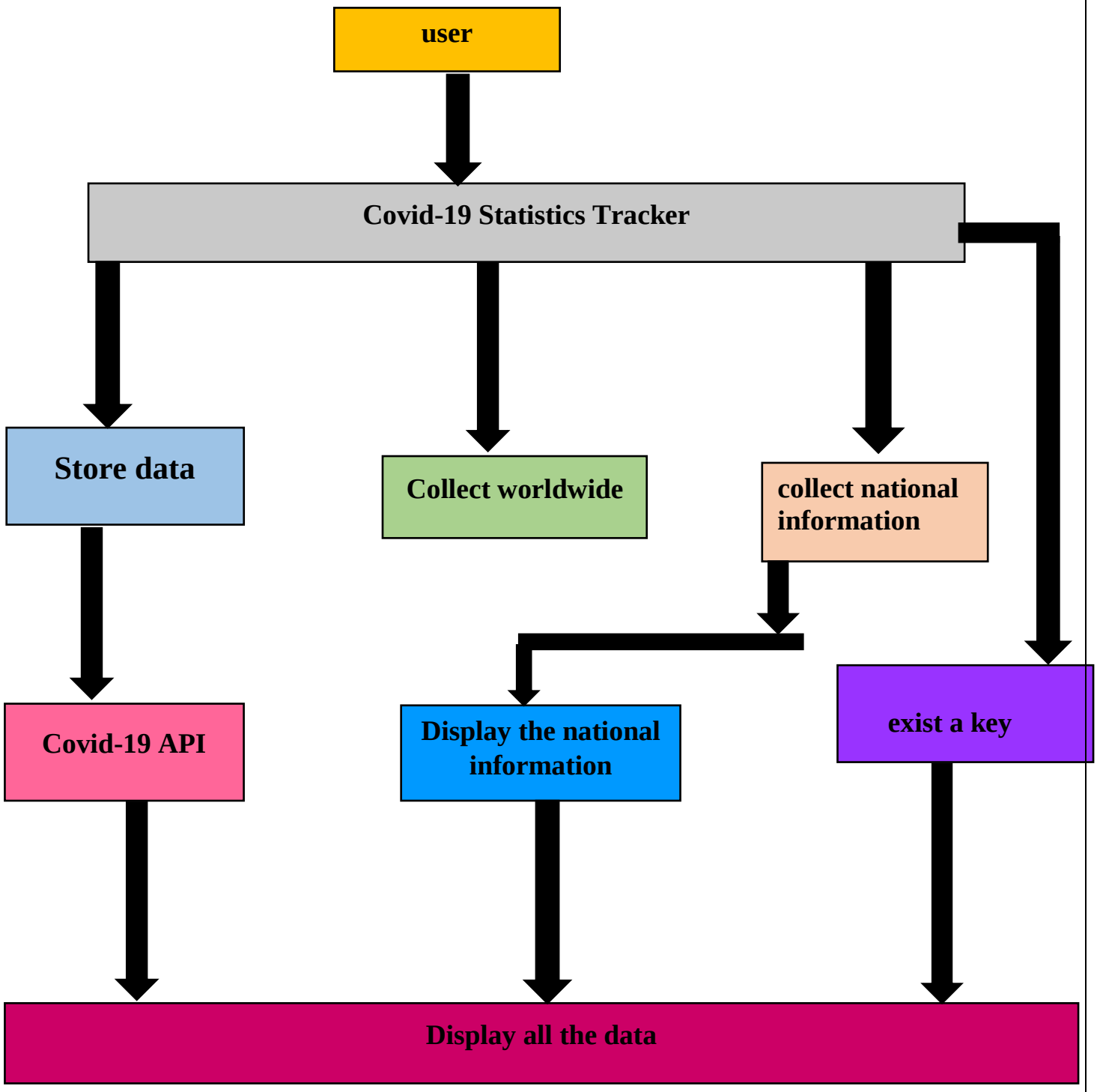
• Documentation of the API integration and the methods used to fetch and display COVID 19 data.

• Explanation of any assumptions made and potential improvements.

Solution:

Real-Time COVID-19 Statistics Tracker

1.Data Flow Diagram



2.Implementation

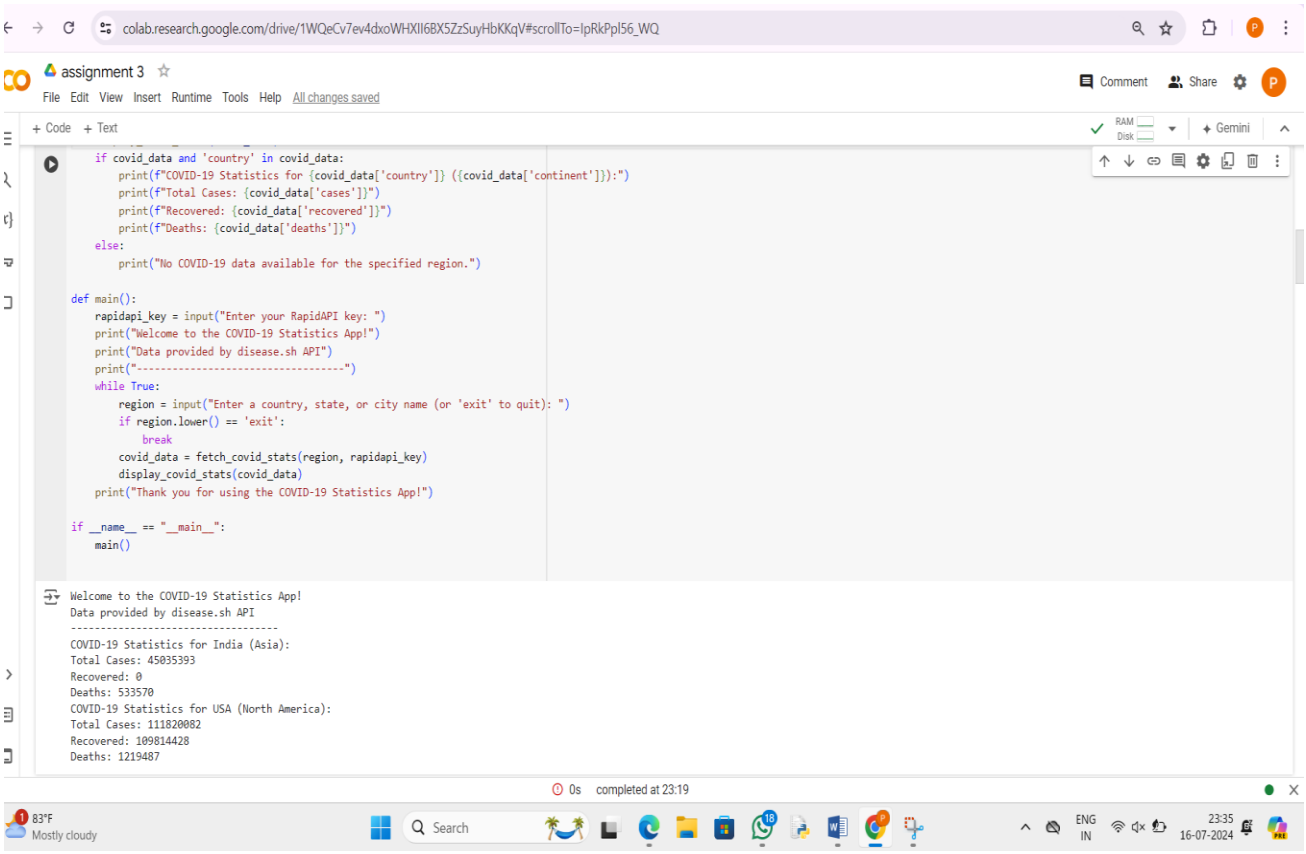
```
import requests

def fetch_covid_stats(region, rapidapi_key):
    url = f"https://disease.sh/v3/covid-19/countries/{region}"
    headers = {
        'x-rapidapi-host': 'disease.sh',
        'x-rapidapi-key': rapidapi_key
    }

    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()  # Raise exception for bad status codes

        covid_data = response.json()
        return covid_data
    except requests.exceptions.RequestException as e:
        print(f"Error fetching COVID-19 data: {e}")
        return None

def display_covid_stats(covid_data):
    if covid_data and 'country' in covid_data:
        print(f"COVID-19 Statistics for {covid_data['country']} ({covid_data['continent']}):")
        print(f"Total Cases: {covid_data['cases']}")
        print(f"Recovered: {covid_data['recovered']}")
        print(f"Deaths: {covid_data['deaths']}")
    else:
        print("No COVID-19 data available for the specified region.")

def main():
    rapidapi_key = input("Enter your RapidAPI key: ")
    print("Welcome to the COVID-19 Statistics App!")
    print("Data provided by disease.sh API")
    print("----------------------------------")
    while True:
        region = input("Enter a country, state, or city name (or 'exit' to quit): ")
        if region.lower() == 'exit':
            break
        covid_data = fetch_covid_stats(region, rapidapi_key)
        display_covid_stats(covid_data)
    print("Thank you for using the COVID-19 Statistics App!")

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

3.Display the Real-Time COVID-19 Statistics Tracker

Welcome to the COVID-19 Statistics App!
Data provided by disease.sh API

COVID-19 Statistics for India (Asia):
Total Cases: 45035393
Recovered: 0
Deaths: 533570
COVID-19 Statistics for USA (North America):
Total Cases: 111820082
Recovered: 109814428
Deaths: 1219487
addCode
addText

4.User Input



5.Documentation:

OBJECTIVES: By providing users precise details, 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 Making Decisions: By providing current data, you may help the general public, healthcare professionals, and lawmakers make well-informed decisions.
Highlights dashboard Overview
Global Statistics: List all instances that have been confirmed worldwide, along with the number of fatalities, recoveries, and active cases. Regional Breakdown: Provide charts and interactive maps with data specific to individual countries or areas. Graphs that show changes in COVID-19 metrics on a daily, weekly, and monthly basis can be used to analyse trends over time.