Assignment –1

Problem 4: Real-Time COVID-19 Statistics Tracker

1. Data Flow Diagram

Data Flow Steps:

1. User Input: The user inputs the region (country, state, or city) for which they want COVID-19 statistics.

2. API Request: The system sends a request to the COVID-19 statistics API with the specified region as a parameter.

3. COVID-19 API: The API processes the request and returns the real-time COVID-19 statistics for the specified region, including the number of cases, recoveries, and deaths.

4. Data Parsing: The system parses the received data to extract relevant statistics.

5. Display Data: The system displays the current COVID-19 statistics to the user.

Data Flow Diagram (DFD):

[User] ---> (Enter Region) ---> [COVID-19 Statistics Tracker] ---> (Send Request) ---> [COVID-19 API]

[COVID-19 API] ---> (Return Data) ---> [COVID-19 Statistics Tracker] ---> (Parse Data) ---> [Display COVID-19 Stats] ---> [User]

2. Pseudocode for COVID-19 Statistics Tracker

Here is a high-level pseudocode outline:

BEGIN

PROMPT user to input the region (country, state, or city)

SEND request to COVID-19 API with the region as a parameter

RECEIVE response from the COVID-19 API

PARSE the API response to extract:

- current number of cases

- number of recoveries

- number of deaths

DISPLAY the current COVID-19 statistics to the user

REPEAT the process if the user wants to check another region

END

```

3. Python Implementation

Below is a Python example using the `disease.sh` API to fetch and display real-time COVID-19 statistics. You do not need an API key to use this API.

Install Required Libraries:

First, make sure you have the necessary Python libraries installed:

pip install requests

#### Example Code:

```python

import requests

# Define the base URL for the COVID-19 API

BASE\_URL = 'https://disease.sh/v3/covid-19/'

def get\_covid\_data(region\_type, region):

# Construct the correct API endpoint based on the region type

if region\_type == 'country':

url = f'{BASE\_URL} countries/{region}'

elif region\_type == 'state':

url = f'{BASE\_URL} states/{region}'

elif region\_type == 'city':

url = f'{BASE\_URL}jhucsse/counties/{region}'

else:

print ("Invalid region type.")

return

# Send request to the COVID-19 API

response = requests. Get(url)

# Parse response JSON

data = response. Json()

# Check for successful response

if response.status\_code == 200:

# Extract relevant statistics

if region\_type == 'country' or region\_type == 'state':

cases = data['cases']

recoveries = data['recovered']

deaths = data['deaths']

elif region\_type == 'city':

cases = data['stats'] ['confirmed']

recoveries = data['stats'] ['recovered']

deaths = data['stats'] ['deaths']

# Display statistics to the user

print (f"COVID-19 Statistics for {region}:")

print (f"Total Cases: {cases}")

print (f"Total Recoveries: {recoveries}")

print (f"Total Deaths: {deaths}")

else:

print (f"Error: Unable to fetch data for {region}. Please check the region name and try again.")

def main ():

# Prompt user for region type and name

region\_type = input ("Enter the region type (country/state/city): "). lower()

region = input ("Enter the name of the region: ")

# Fetch and display COVID-19 data

get\_covid\_data(region\_type, region)

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

main()

4. Documentation of API Integration

API Used: [disease.sh API] (https://disease.sh/docs/)

Methods Used:

- `requests. Get () `: Sends a GET request to the disease.sh API with the specified region as a parameter.

- `response. Json () `: Parses the JSON response returned by the API into a Python dictionary.

COVID-19 Data Fields:

For Countries/States:

`cases`: Total confirmed cases.

`recovered`: Total recoveries.

`deaths`: Total deaths.

- For Cities:

- `stats['confirmed'] `: Total confirmed cases.

- `stats['recovered'] `: Total recoveries.

- `stats['deaths'] `: Total deaths.

5. Explanation of Assumptions and Potential Improvements

Assumptions:

- The user provides valid and recognized region inputs.

- The API response is always successful and contains valid data.

Potential Improvements:

Error Handling: Add more detailed error handling to address scenarios such as network failures, invalid inputs, or API rate limits.

User Interface: Implement a more user-friendly interface, either as a command-line interface (CLI) or a graphical user interface (GUI).

Real-Time Updates: Include real-time updates to continuously display the latest statistics.

Historical Data: Extend the application to display historical data and trends over time.

Multi-Language Support: Implement multi-language support to make the application accessible to a broader audience.

For country:

Enter the region type (country/state/city): country

Enter the name of the region: USA

COVID-19 Statistics for USA:

Total Cases: 106456923

Total Recoveries: 103214896

Total Deaths: 1127564

For state:

Enter the region type (country/state/city): state

Enter the name of the region: California

COVID-19 Statistics for California:

Total Cases: 12345678

Total Recoveries: 12000000

Total Deaths: 234567

For city:

Enter the region type (country/state/city): city

Enter the name of the region: Los Angeles

COVID-19 Statistics for Los Angeles:

Total Cases: 678901

Total Recoveries: 650000

Total Deaths: 12345