**Steps to Run the App:**

1. In the folder in which file rest\_api.py is present, enter cmd to open command prompt from that folder.
2. Enter following in command prompt: **docker-compose up**
3. The server will start running after some time. Don’t close this command prompt.
4. Open another command prompt from same folder. In this command prompt enter following command: **python DataCollection.py**
5. After few seconds the first result will be displayed on command prompt. Press Enter again and again to see all results from GET request.

**My approach to complete the task:**

1. **Data Collection From COVID-19 Servers:**

I Send Get requests for following three online API servers:

1. <https://covid-api.com/api/>
2. <https://corona-virus-stats.herokuapp.com/api/v1/cases/>
3. <https://covid19-stats-api.herokuapp.com/api/v1/cases>

Data of following is obtained by GET Request as response: Region Names, Provinces BY ISO, List Of Reports, Total data By Date, General Stats, Data for one Country.

Following useful data is selected for sending as to Flask REST API:

1. Confirmed, Deaths, and Recovered Cases of a single country.
2. General Stats which include Total Cases in world, Total recovered in world, Total deaths in world and Total currently infected in world.
3. Total Cases on a given date. Send the date and get following: Confirmed Cases, Deaths, Recovered and Active Cases on that particular given date.
4. **Flask RestFul API:**

First all requirements from requirements.txt was installed by pip install -r requirements.txt.

Flask, API, SQLAlchemy objects created.

In end added, app.run(debug=True).

3 models are created by 3 classes: TotalDayModel, GeneralStatsModel and DataDateModel. In all of these models classes columns with required datatypes are created.

db.create\_all() in the end of models to create database.

reqparse.RequestParser() is used to make put\_args and update\_args for all of three models.

Resource Fields Dictionaries are created for all three models.

We created other three classes for each Database model which include get, put and patch methods in them.

Get method is made to return json object results as response. Put method inserts new json records in our database. Patch method is used to update our existing json records.

Right before get, put and patch methods we added @marshal\_with(resource\_fields) lines.

In get if no results could be obtained for given primary key then app shows could not find data with that key. If primary key already exists in put method then app shows that primary key already is present.

db.session.commit() is used in put and patch to insert new data.

api.add\_resource(ClassName,’Route url’) is used for all three classes.

To test our restful app, we added following in the end of DataCollection.py:

BASE="http://127.0.0.1:5000/"

response=requests.put(BASE+"day/"+"China",dataOneCountry)

print(response.json())

input()

response=requests.get(BASE+"day/"+"China")

print(response.json())

response1=requests.put(BASE+"stats/"+"1",GeneralStats)

print(response1.json())

input()

response1=requests.get(BASE+"stats/"+"1")

print(response1.json())

response2=requests.put(BASE+"datedata/"+"2020-04-15",Data\_Date)

print(response2.json())

input()

response=requests.get(BASE+"datedata/"+"2020-04-15")

print(response2.json())

1. **Containarized app using docker and docker-compose:**

Downloaded and installed docker desktop.

Added Dockerfile in folder.

Added docker-compose.yaml file in folder.

In rest\_api.py , I added in app.run(host="0.0.0.0").

In command prompt (from folder where all files are present) typed command docker-compose up. After few minutes in downloaded all files and our app is containerized in few minutes.

Another command prompt was opened from same folder. I entered python DataCollection.py and all responses could be seen on command prompt.