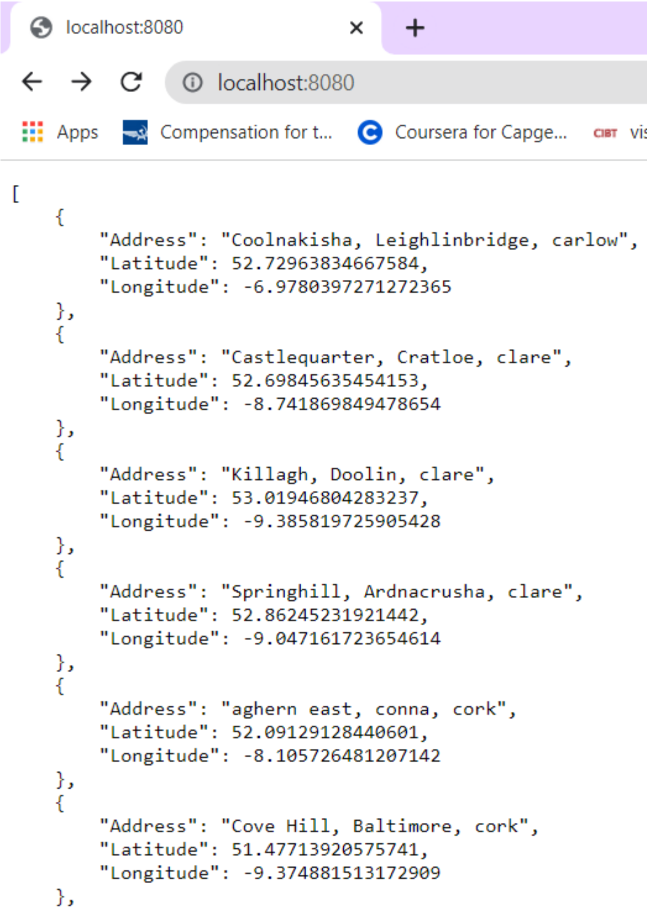
Approach ➔

I have developed the geocoder using ATTD (Acceptance test driven development). I have structured the project in such a way that my controller, service, and data structure layers are different. Since there is no real-time database used in this project, I have used pandas dataframe as an in-memory database. I am loading the townlands and county csv data in this dataframe on application startup. Hence, approach for loading this dataframe is to load once and re-use for all API calls. I have also tried to make most of the methods within the class as private in order to encapsulate the business logic.

Applications starts from main.py which then creates an object of GetCoordinates class. The constructor of this class auto triggers methods of PrepareDatabase class to initialise the required dataframes. I have also created two controllers to listen on port 8080. Below is the path of the controllers:

1) GET on http://localhost:8080/
2) POST on http://localhost:8080/

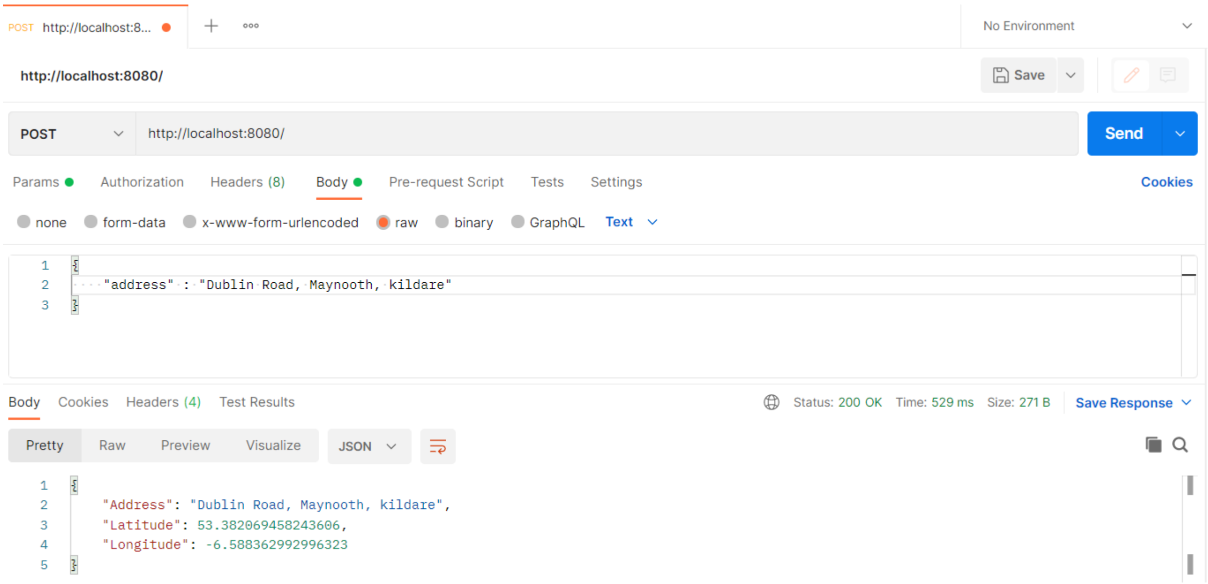
For the GET method of above link, the application runs itself fetches the data for address from addresses_for_task.csv file and return the response in the JSON format. The figure 1 shows the output for the same.



```
[
    {
        "Address": "Coolnakisha, Leighlinbridge, carlow",
        "Latitude": 52.72963834667584,
        "Longitude": -6.9780397271272365
    },
    {
        "Address": "Castlequarter, Cratloe, clare",
        "Latitude": 52.69845635454153,
        "Longitude": -8.741869849478654
    },
    {
        "Address": "Killagh, Doolin, clare",
        "Latitude": 53.01946804283237,
        "Longitude": -9.385819725905428
    },
    {
        "Address": "Springhill, Ardnacrusha, clare",
        "Latitude": 52.86245231921442,
        "Longitude": -9.047161723654614
    },
    {
        "Address": "aghern east, conna, cork",
        "Latitude": 52.09129128440601,
        "Longitude": -8.105726481207142
    },
    {
        "Address": "Cove Hill, Baltimore, cork",
        "Latitude": 51.47713920575741,
        "Longitude": -9.374881513172909
    },
```

Figure 1

For the POST method of above link, the application requires an input parameter of type string and key as ‘address’. It accepts the address, performs the calculation, and returns the GPS co-ordinates of the given address in the JSON format. The figure 2 shows the output for the same.



http://localhost:8080/

POST http://localhost:8080/

```
{
    "address" : "Dublin Road, Maynooth, kildare"
}
```

Status: 200 OK Time: 529 ms Size: 271 B

```
{
    "Address": "Dublin Road, Maynooth, kildare",
    "Latitude": 53.382069458243606,
    "Longitude": -6.588362992996323
}
```

Figure 2

Advantages ➔

1) The two datasets (townlands and county) contain only the required fields and the other columns from CSV file is dropped in order to save the processing time further during the endpoint calls.
2) The two dataframes are used as in-memory database and are initialized only once and reused for each endpoint calls.
3) Since dataframes are used as in-memory database, a lot of processing time in connecting to database and retrieving the required information is saved.
4) The program is divided into 3 layers due to which the controller and the logical layer are separate and easy to modify.

Disadvantages ➔

1) As the database volume increases, a lot of local memory will be consumed. Hence, data needs to be stored in database.
2) When the application restarts the dataframes are required to be initialized again. This can be avoided by using a real-time database like PostgreSQL or Oracle.