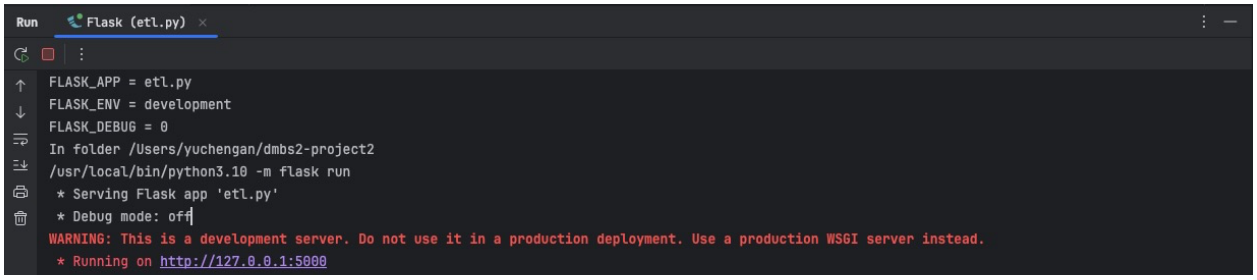
Project 2 Report and Screenshots

Name: Project Part 2 - Building a Data Lake and Data Warehouse

Teams: Yucheng An, Henglay Eung, Harsh Umesh Bhanushali (GitHub Repo Folder: Henglay-Yucheng-Harsh)

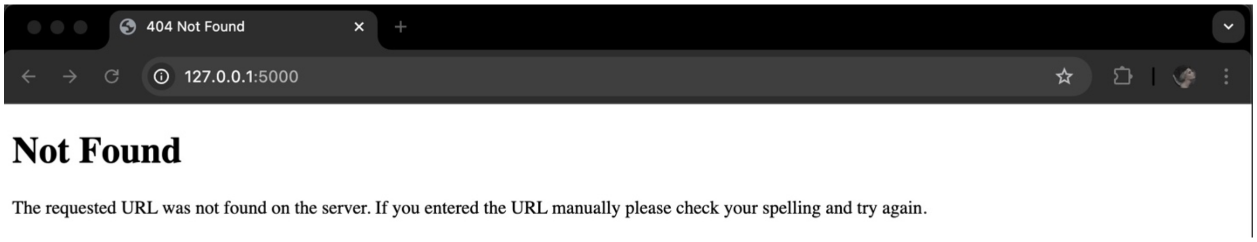
(Pre-prepare: create teamwork repo, put etl.py into repo, assigned tasks for each)

First, we downloaded the etl.py and installed the related package as required. We followed the tips from the console when we ran etl.py file.



```
FLASK_APP = etl.py
FLASK_ENV = development
FLASK_DEBUG = 0
In folder /Users/yuchengan/dmbs2-project2
/usr/local/bin/python3.10 -m flask run
 * Serving Flask app 'etl.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
```

After we visited http://127.0.0.1:5000/

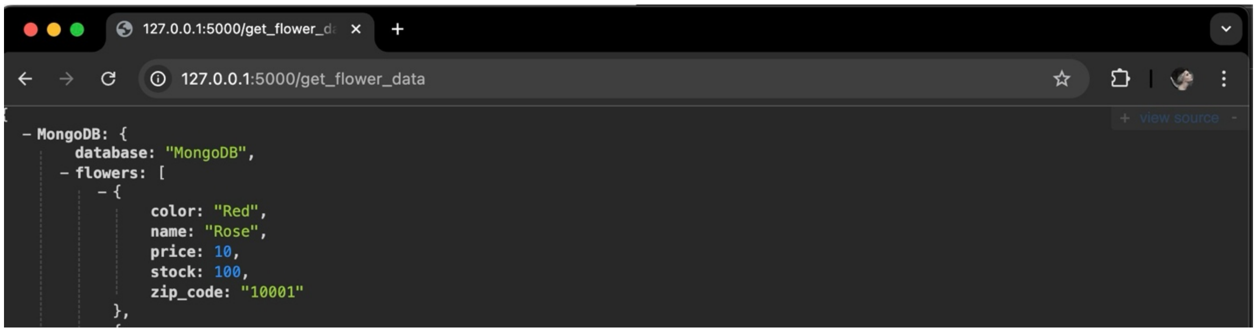


Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Seem to have started successfully. Next, we tried to access http://127.0.0.1:5000/get_flower_data

I use the JSON extension to make JSON data have a better structure to view in the browser.



```
{
- MongoDB: {
      database: "MongoDB",
    - flowers: [
        - {
              color: "Red",
              name: "Rose",
              price: 10,
              stock: 100,
              zip_code: "10001"
          },
```

Task 1

Every default data has been successfully accessed by using the Get method. **Finished Task 1.**

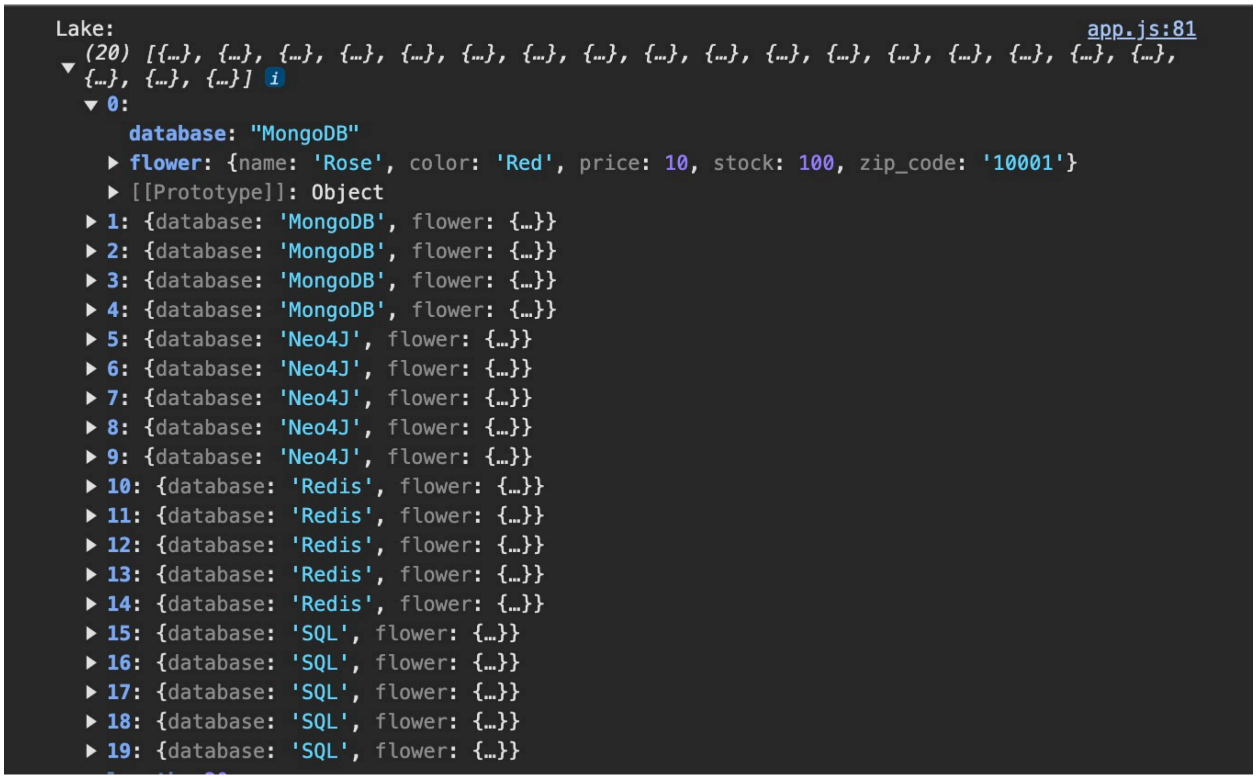
Second, we wrote JavaScript named “app.js” to deal with the data extraction process. We successfully store the data into an array named “Lake”.

We followed the instructions as:

- *Each object shall include its database source (e.g. "db":"sql")*

- *Each object shall only store one flower object value*

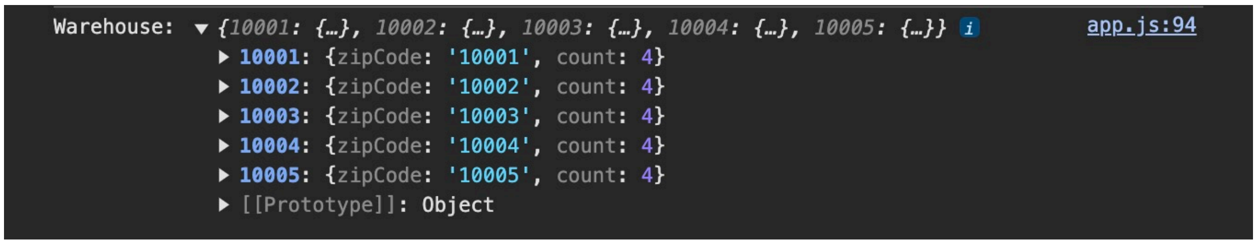
The only difference is we changed. "db":"sql" to “database”: “SQL” for better view. **Finished Task 2.**



```
Lake:                                                                        app.js:81
▼ (20) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
  {…}, {…}, {…}] i
  ▼ 0:
      database: "MongoDB"
    ▶ flower: {name: 'Rose', color: 'Red', price: 10, stock: 100, zip_code: '10001'}
    ▶ [[Prototype]]: Object
  ▶ 1: {database: 'MongoDB', flower: {…}}
  ▶ 2: {database: 'MongoDB', flower: {…}}
  ▶ 3: {database: 'MongoDB', flower: {…}}
  ▶ 4: {database: 'MongoDB', flower: {…}}
  ▶ 5: {database: 'Neo4J', flower: {…}}
  ▶ 6: {database: 'Neo4J', flower: {…}}
  ▶ 7: {database: 'Neo4J', flower: {…}}
  ▶ 8: {database: 'Neo4J', flower: {…}}
  ▶ 9: {database: 'Neo4J', flower: {…}}
  ▶ 10: {database: 'Redis', flower: {…}}
  ▶ 11: {database: 'Redis', flower: {…}}
  ▶ 12: {database: 'Redis', flower: {…}}
  ▶ 13: {database: 'Redis', flower: {…}}
  ▶ 14: {database: 'Redis', flower: {…}}
  ▶ 15: {database: 'SQL', flower: {…}}
  ▶ 16: {database: 'SQL', flower: {…}}
  ▶ 17: {database: 'SQL', flower: {…}}
  ▶ 18: {database: 'SQL', flower: {…}}
  ▶ 19: {database: 'SQL', flower: {…}}
```

Task 2

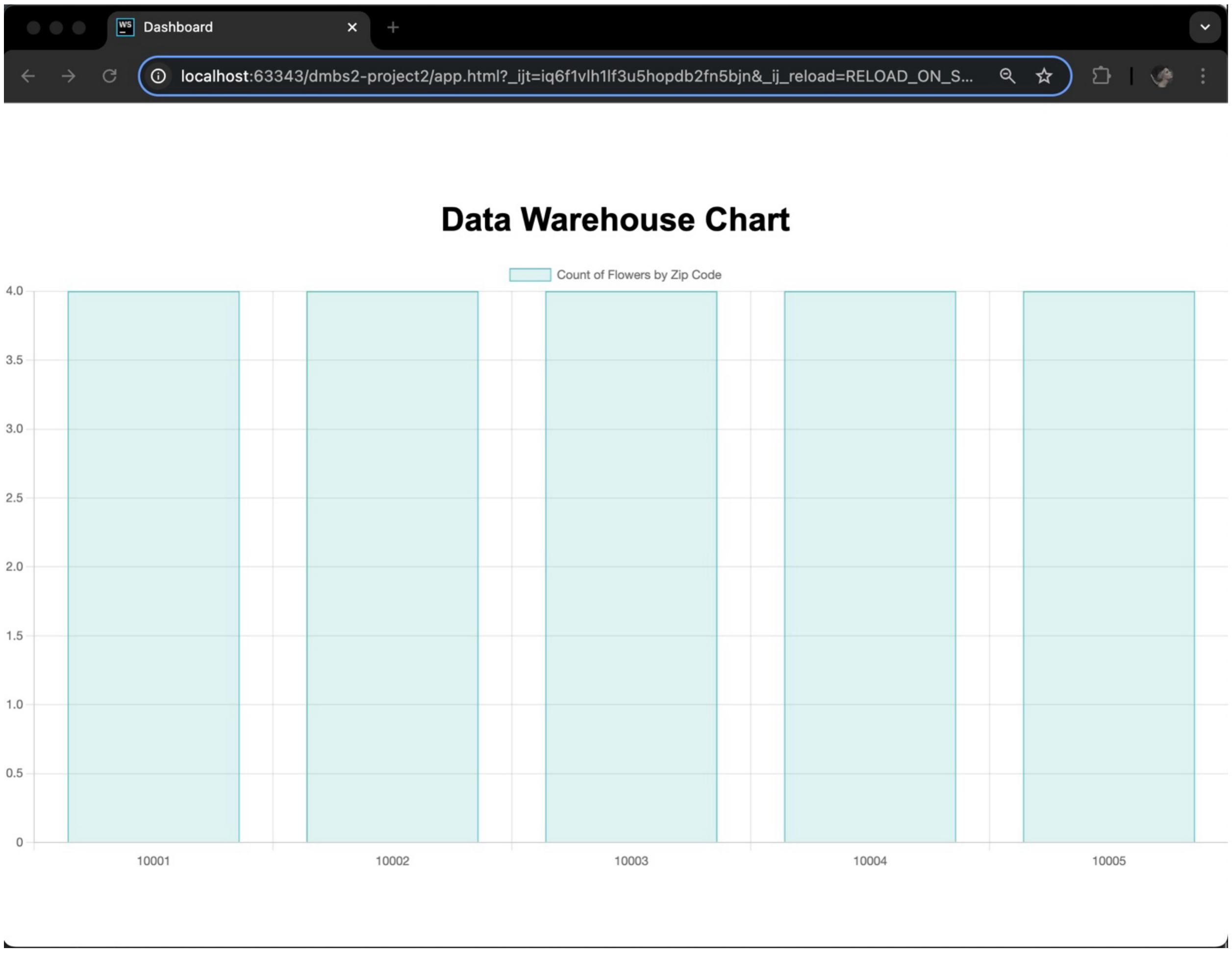
Next, we started dealing with Data Transformation, we used *Map-Filter-Reduce query strings* to query the information that we might need. We stored the data in the “Warehouse” for later usage. **Finished Task 3.**



```
Warehouse:  ▼ {10001: {…}, 10002: {…}, 10003: {…}, 10004: {…}, 10005: {…}} i      app.js:94
              ▶ 10001: {zipCode: '10001', count: 4}
              ▶ 10002: {zipCode: '10002', count: 4}
              ▶ 10003: {zipCode: '10003', count: 4}
              ▶ 10004: {zipCode: '10004', count: 4}
              ▶ 10005: {zipCode: '10005', count: 4}
              ▶ [[Prototype]]: Object
```

Task 3

Now, we already have the data that we want to display in the dashboard. Started working on the Data loading part. We created the <Canvas> label to display the bar chart. The X-axis presents the zip code, and the Y-axis presents the total number of flowers in each zip code. We adjusted the view central and title. **Finished Task 4.**



Task 4

All code files have been pushed to the following GitHub repo:
https://github.com/SE4CPS/dms2/tree/main/projects/project-2/Henglay-Yucheng-Harsh

Using an SQL database instead of a local runtime variable significantly impacts a Data Lake or Data Warehouse's design and functionality such as ETL or ELT efficiency, the local runtime variable requires manual coding for extraction, transformation, and loadings. But SQL will integrate with ETL tools and set up automatic workflow for data transformation. Partial loading will significantly increase the performance of the data engineering process, especially for the big data set. In this project, we used *etl.py* to simulate the environment that collects data from multiple data sources including MongoDB, Neo4J, Redis, and SQL) The API port has been designed well. We used the real example to simulate the right process of ETL and learned what these three words (Extraction, Transformation, Loading) need to do in data engineering. We also learned about the relationship between the lake and the warehouse. Our team worked well during Project 2, we all were assigned the different parts of this project and pushed our group repo.