Name: **Dhruv Maheshbhai Patel**  
Student ID: **9062297**  
Subject: **Database Automation**

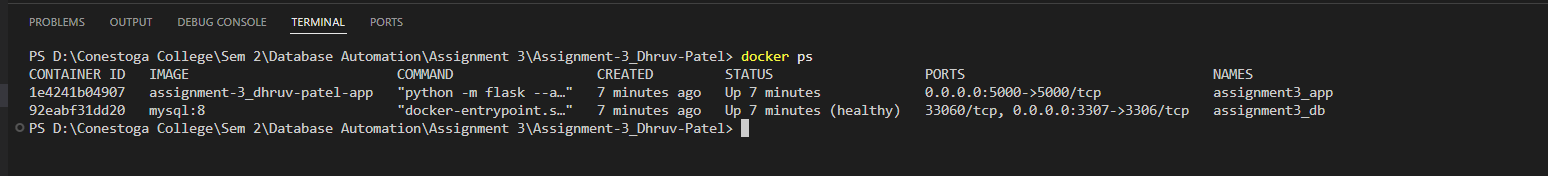
**Assignment Report 2**

In this project, a portion of NYC 311 service requests is extracted, processed as data, loaded into a MySQL database, and a web application is provided to visualize the count of complaints by each borough. Selenium is automated and used to test Docker and is available and reproducible.

**Dataset** Dataset: NYC 311 Service Requests

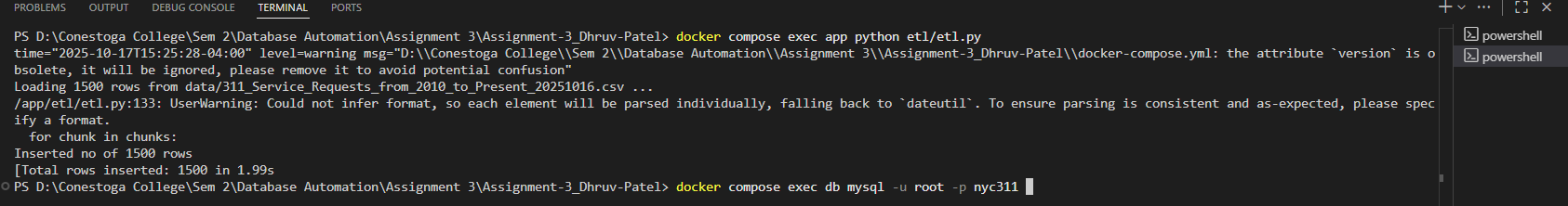
- Slice used for testing: `data/fixture.csv` (50 rows)

**Step 1 – Setup:**

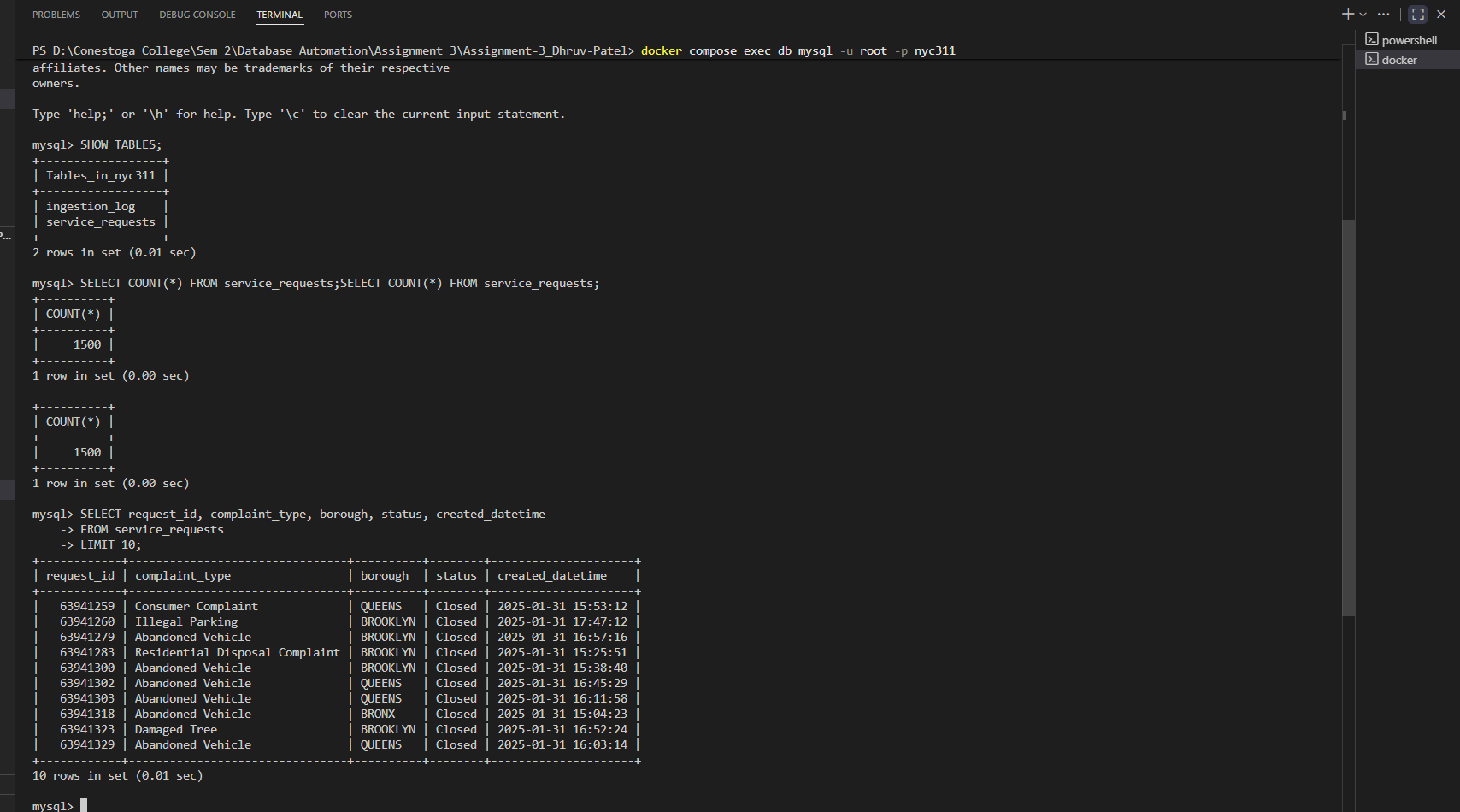


**Step 2 – ETL (Extract, Transform, Load)**

The screenshot depicts the ETL script in operation and loading a slice of the dataset into the service\_requests table. The resulting output of the script validates the quantity of rows that have been added, and it also illustrates that the script can clean, convert and import CSV data to MySQL.

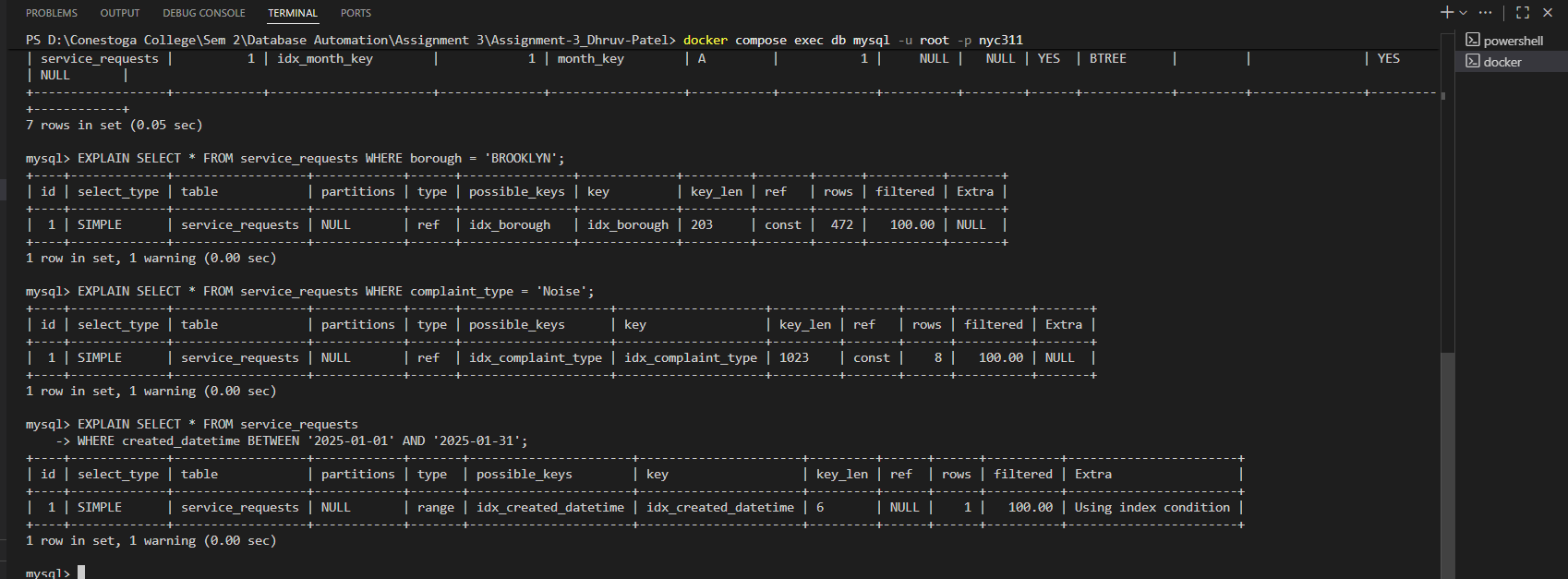


**Step 3 – Database Schema & Indexes**

This is a screenshot of the schema of the service requests table with column names, data type, and indexes. It confirms that the database format matches the ETL process cleaned and transformed data guaranteeing the efficiency of storage and retrieval.

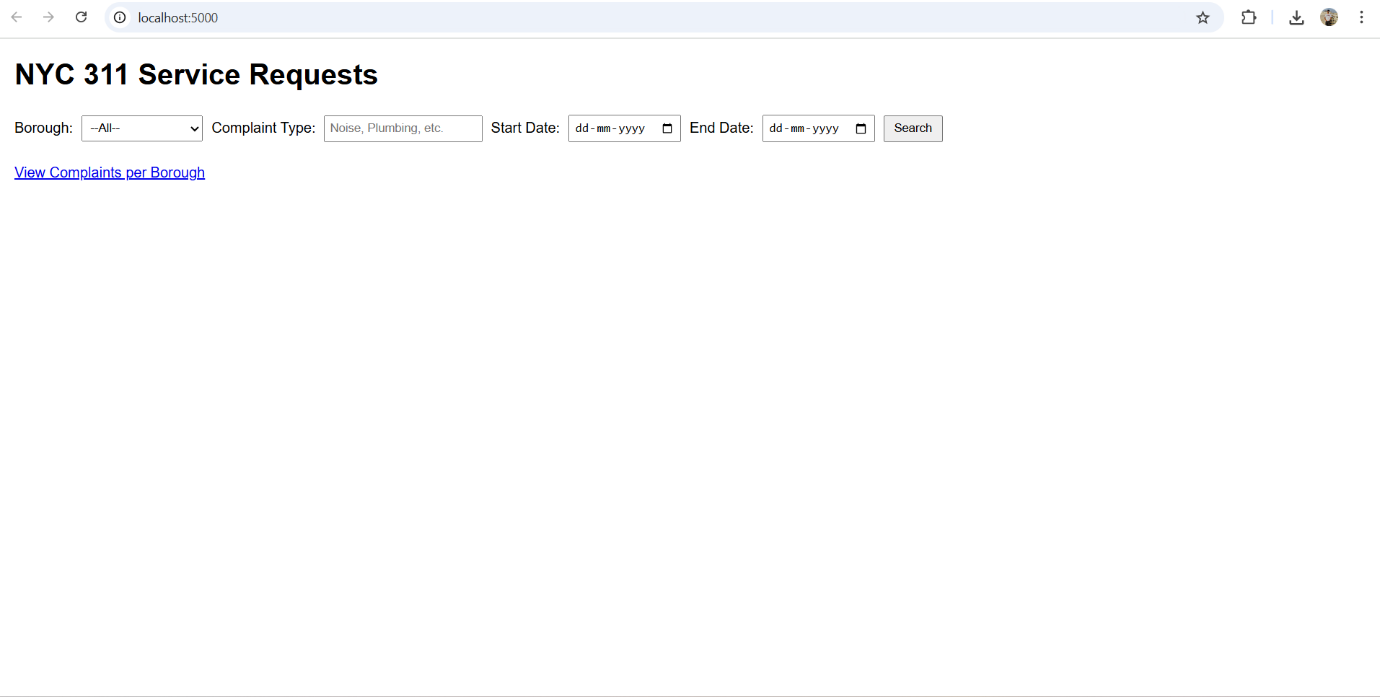
**Step 3 – Database Schema & Indexes**

Explain indexes for BROOKLYN, Noise, Date:

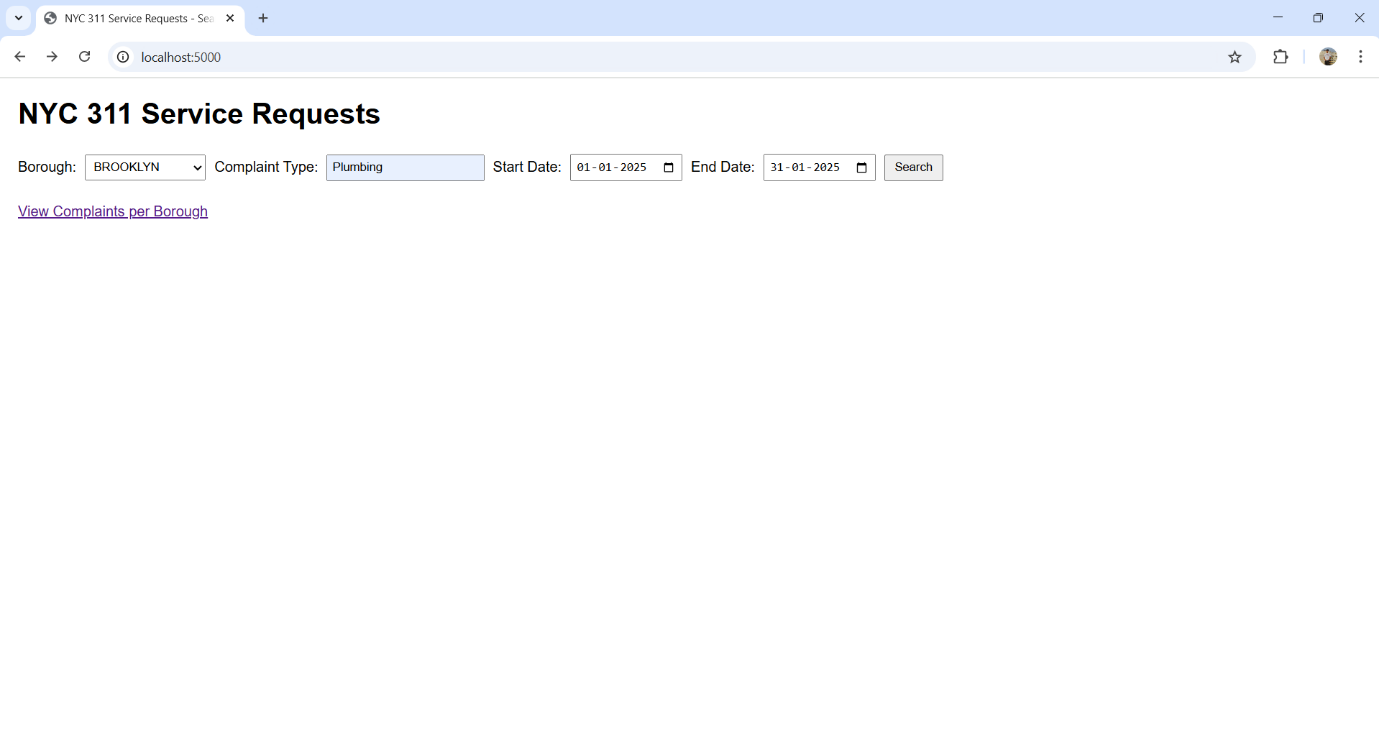


**Step 4 – Web Application:**

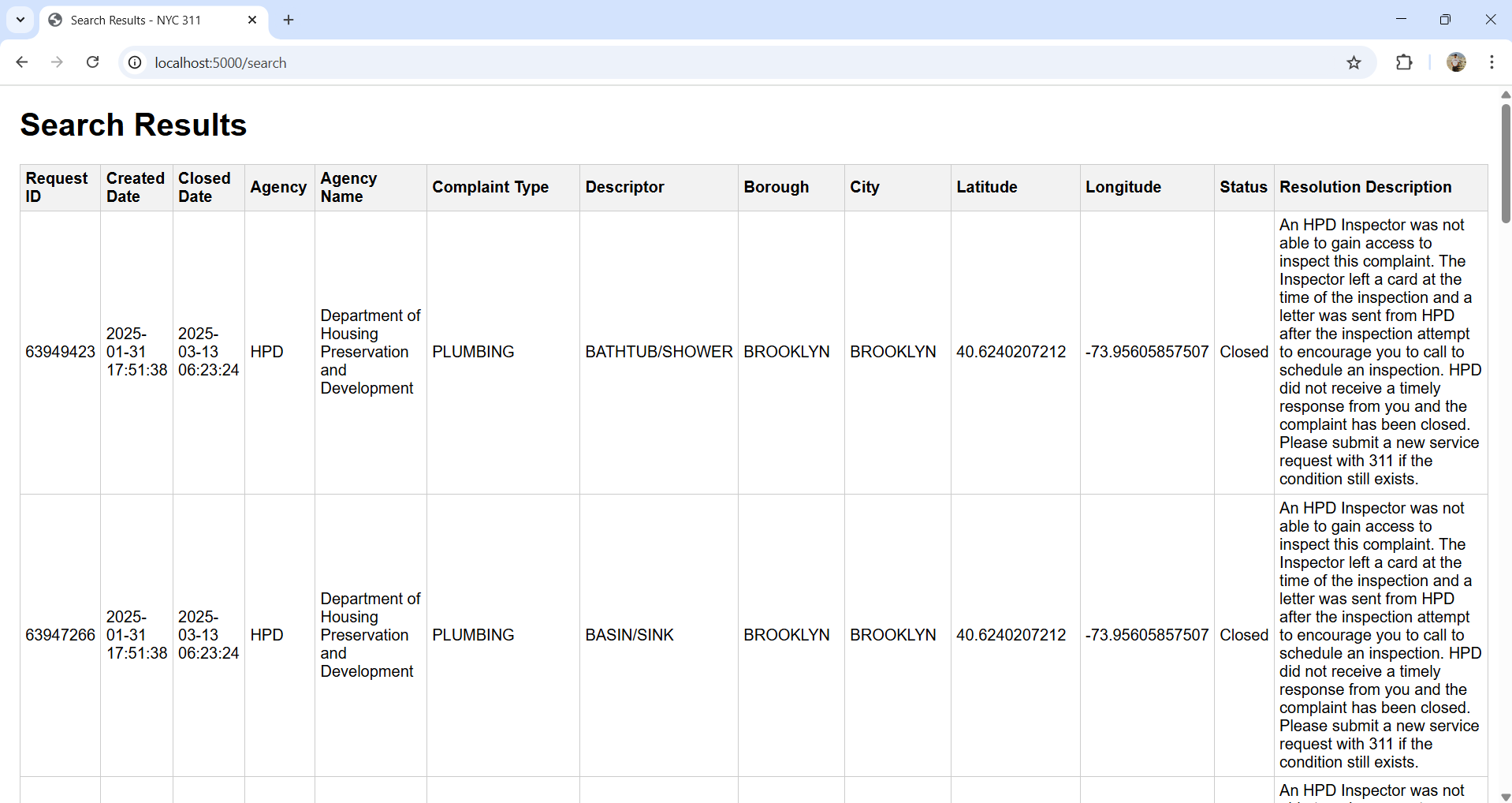
This is a screenshot of the aggregate page of the web application, which contains the number of complaints in each borough. It confirms that the data that was loaded using ETL is available in the web interface and the aggregation query itself functions as expected.



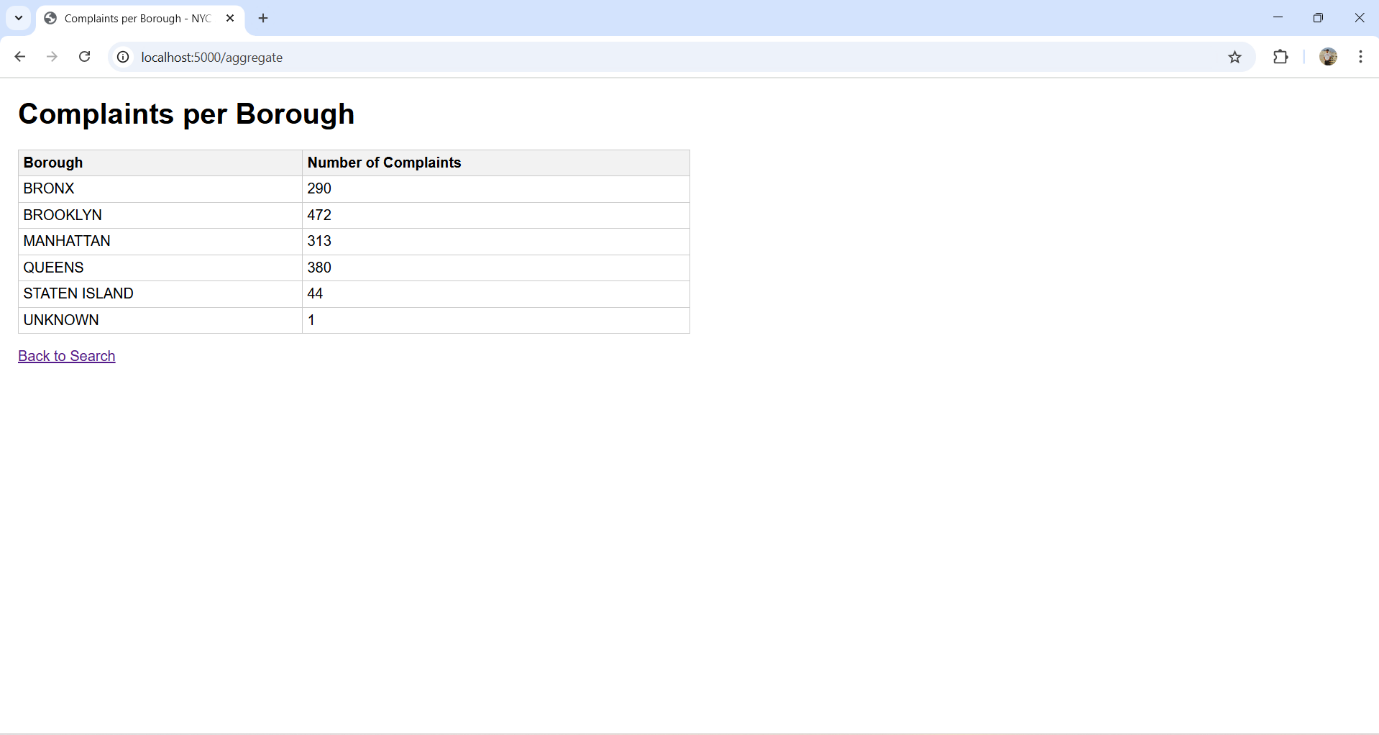
A search form of BROOKLYN and Complaint of Plumbing from 01-01-2025 to 31-01-2025:



Results from MySQL:

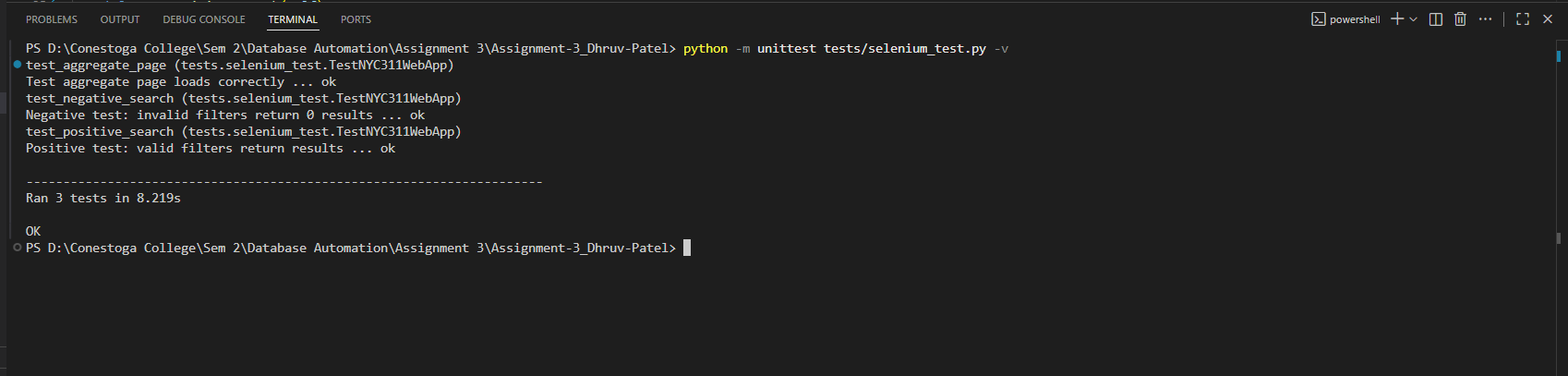


An aggregate view (e.g., complaints per borough):

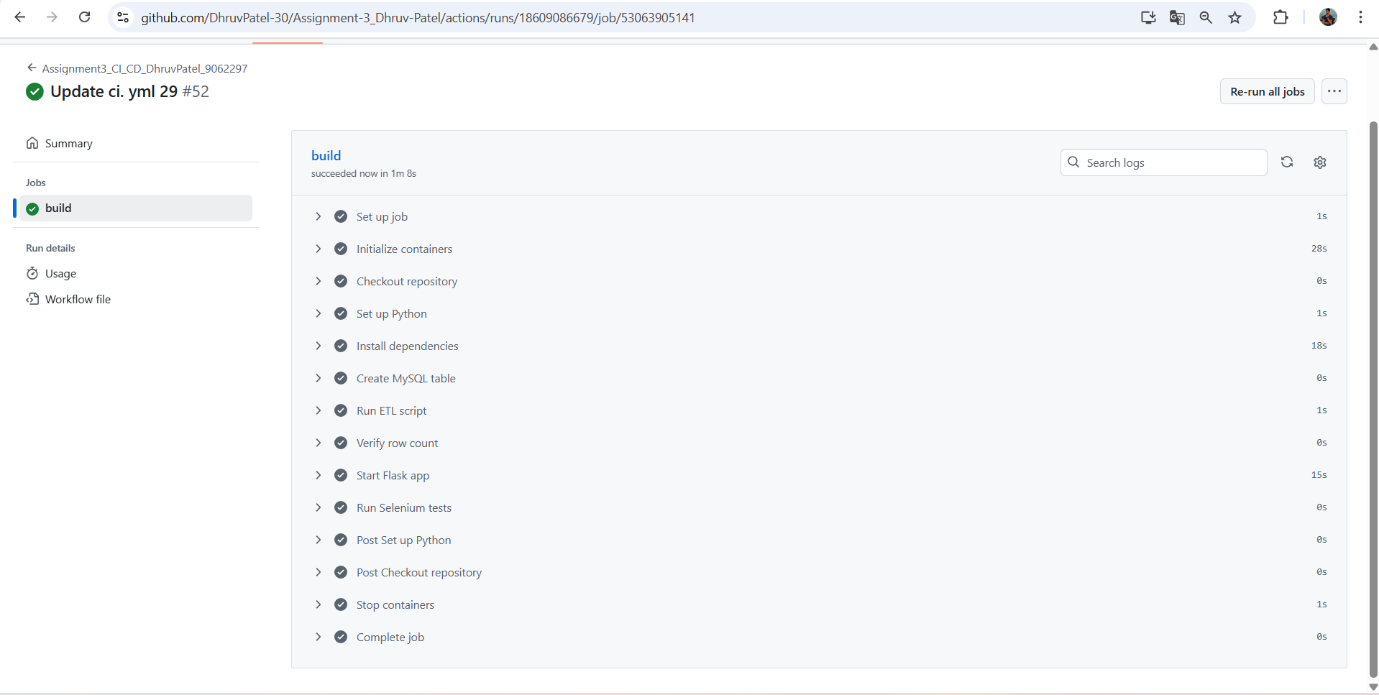


**Step 5 – Automated Testing**

This screenshot shows the automated testing of Selenium and pytest. It verifies the web application pages have loaded properly and that the expected items, e.g. tables, actually exist. This makes the functionality of the web app right.



**Step 6 – CI/CD (GitHub Actions)**



**Reflections:**

* Docker Compose made setting up of the environment easier. The restriction of dependencies and versions served to ensure that the ETL, database, and web app were compatible and did not fail in various machines. One of the problems was to learn the right service connection between app and db containers.
* A chunked ETL pipeline was written to deal with large amounts of data. Ensuring correct conversion of the type and correcting missing or bad data was also a challenge especially date and numeric columns.
* The choice of appropriate data types and a primary key of the service requests table allowed quick inserts and simple access to the results. One of the difficulties was to deal with duplicate rows safely with on duplicate KEY UPDATE.
* The flow of the application was confirmed by doing automated testing with Selenium and pytest. One obstacle was ensuring that the tests were able to locate elements in a reliable way and also running the tests in the headless mode in CI/CD pipelines.

Repo link: https://github.com/DhruvPatel-30/Assignment-3\_Dhruv-Patel/