

Twitter Tweets Data Streaming

Written by : Isabel Johnson

Twitter Tweets Data Streaming

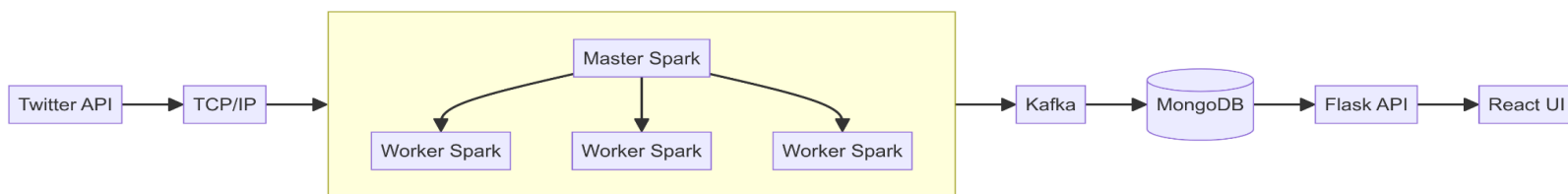
1. Introduction

In this submission, I present a solution designed to manage and analyze large datasets of Twitter data, specifically focusing on tweets about Britney Spears. The task involves

processing TSV file, one approximately 50MB in size, with the option to choose based on system capabilities. The implemented system is crafted to ingest the data efficiently, store it in a queryable format, and provide insightful analytics based on user-defined search terms such as "music." This includes analyses include top 10 tweetsbased on the search term.

2. Design

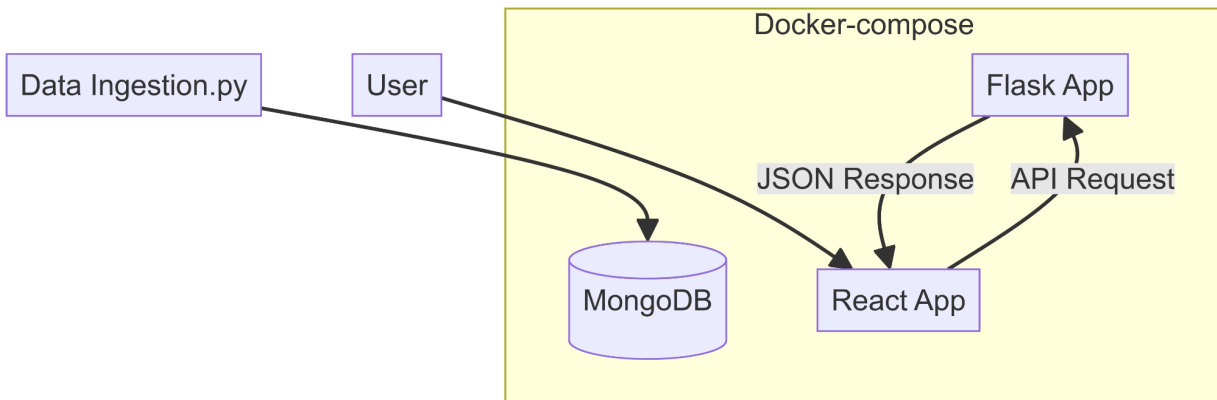
2.1. Enterprise-Level Design for Real-Time Twitter Data Processing



The architecture diagram outlines a real-time data processing and visualization system using various technologies. Spark efficiently handles large-scale, real-time data (best for large dataset. For example :500MB files), Kafka manages high throughput and reliable data flow between systems, and TCP/IP ensures accurate and orderly data transmission from the Twitter API. It begins with data ingestion from the Twitter API, which streams tweets in real-time via TCP/IP to a cluster of Spark nodes (one master and multiple workers) for processing. Post-processing, the data is published to Apache Kafka, which serves as a reliable messaging queue to manage data flow. The processed data is then stored in MongoDB, a NoSQL database that supports high performance and scalability. A Flask API retrieves this data for a front-end application built with React, enabling dynamic data visualization and interaction. This setup is designed to handle large volumes of data efficiently and is ideal for applications that require live data feeds and responsive interfaces.

2.2. Current Design

Since we have the twitter dataset already available. We can go small -scale design.



The diagram represents a web application architecture integrating Python, Flask, MongoDB, React, and Docker-Compose. A Python script, `Data Ingestion.py`, processes and stores data into MongoDB, a NoSQL database, suitable for handling flexible schemas. The user interacts with the application via a React frontend, which communicates with a Flask backend via API requests. The backend retrieves data from MongoDB, processes it, and returns responses in JSON format to the React app. Docker-compose is utilized to manage and orchestrate these components, ensuring the application is scalable and modular. This architecture highlights a clear separation of concerns with an efficient connection between the user interface, data handling, and service management.

2.2.1 Components and design choices

1. **MongoDB:** MongoDB, a NoSQL database, is chosen for its schema flexibility, making it ideal for handling the varied and unstructured data from Twitter. Its dynamic schema capability allows for easy modifications and adaptations as the data or requirements change.
 - Data is stored in the database `Twitter_Tweets` under the collection `Tweets`.
 - Connection credentials are stored in the `config.py` file in the `config` folder.
2. **Data Ingestion:** The data ingestion script (`data_ingest_twitter.py`) is designed to be straightforward and efficient, directly storing data into MongoDB. The data is read from dataset into a dataframe using pandas and pushed to mongodb. Spark is best to load load dataset for example (500 MB). I initially used pyspark to load the dataset instead of pandas. But it needs to manually paste the mongo db spark connector to push data directly. This will add the initial setup complexity while deployment.
3. **React App:** React is chosen for the frontend due to its component-based architecture, which allows for efficient management of state and UI, particularly useful in dynamic, real-time data interactions like searching and displaying Twitter data. React's ability to handle high-volume, dynamic data through efficient DOM updates optimizes performance and user experience.
4. **Flask App:** Flask, a lightweight Python web framework, is employed to serve as the backend API. It offers ease of use and is highly effective for creating RESTful services

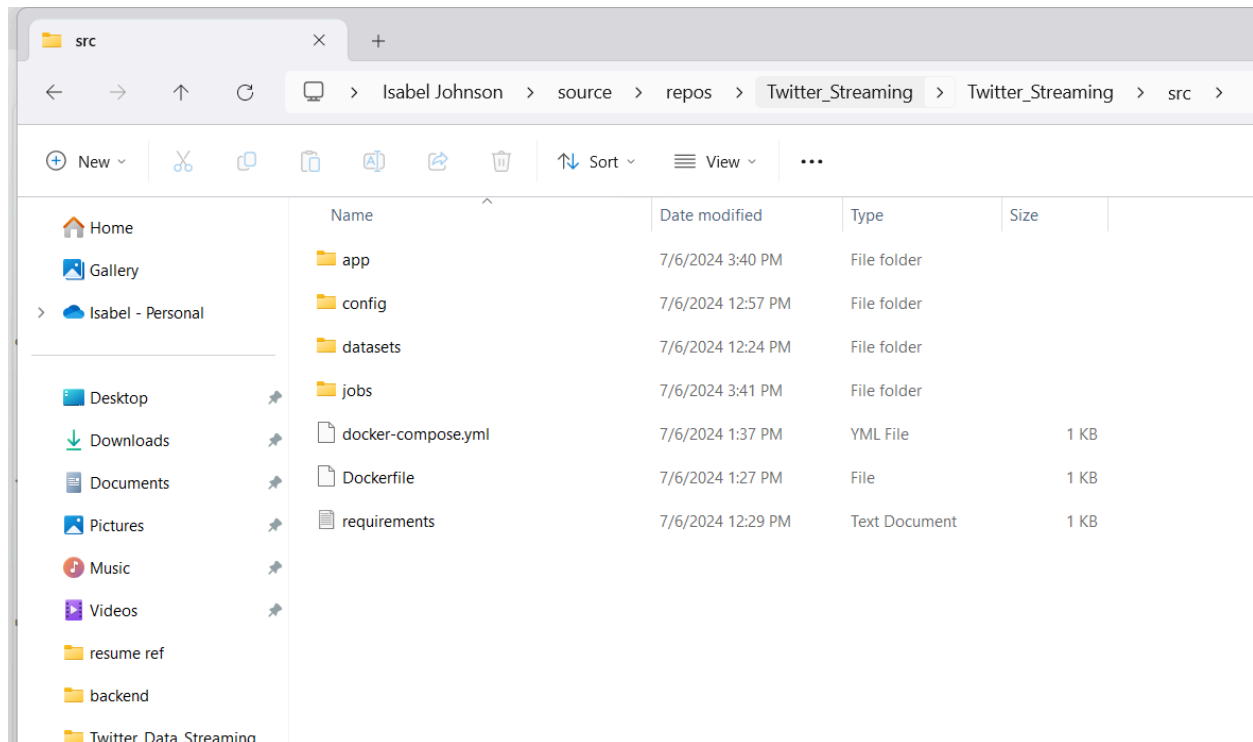
that the React application can consume. Flask's minimalistic and modular design is perfect for small-scale applications, ensuring quick setup and low overhead.

2.3 Deployment files

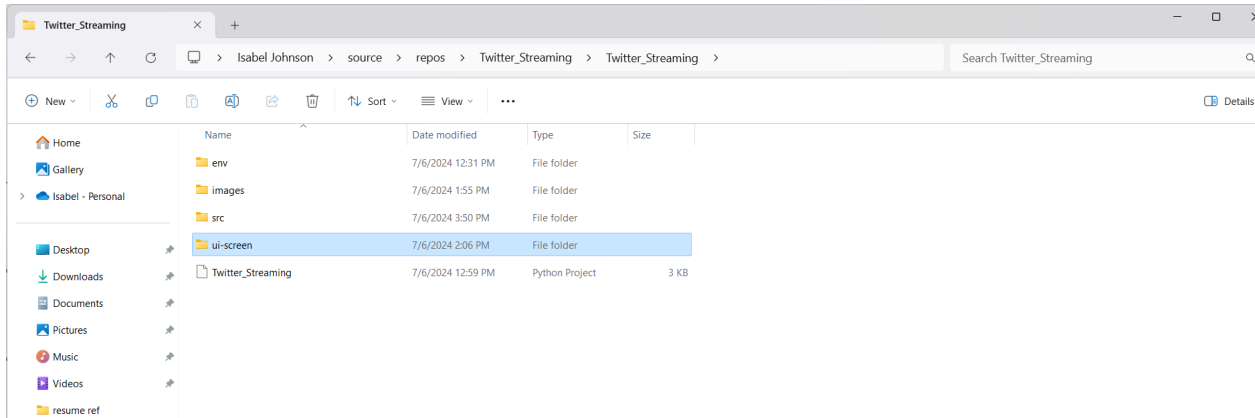
The files are available at this location. Please download and extract to folder name- Twitter_Streaming.

https://drive.google.com/drive/folders/1QCulqHvrnxKMjyrfFisIRztZ9WxezQIY?usp=drive_link

2.3.4 Folder Structure



- **app**: Contains all the files related to the Flask app.
- **config**: Contains connection string details to connect to the database.
- **datasets**: Contains files to be uploaded.
- **jobs**: Contains the data ingestion code file.
- **Docker-Compose.yml**: Contains the code to set up MongoDB.
- **requirements.txt**: Lists all the libraries to be installed.



- **ui-screen:** Contains code files to launch the UI screen.

2.4 Setup

2.4.1 Prerequisites

- Ensure Docker software is installed on your local machine.
- Install Docker from the Docker Documentation. from [Install Docker Desktop](#)

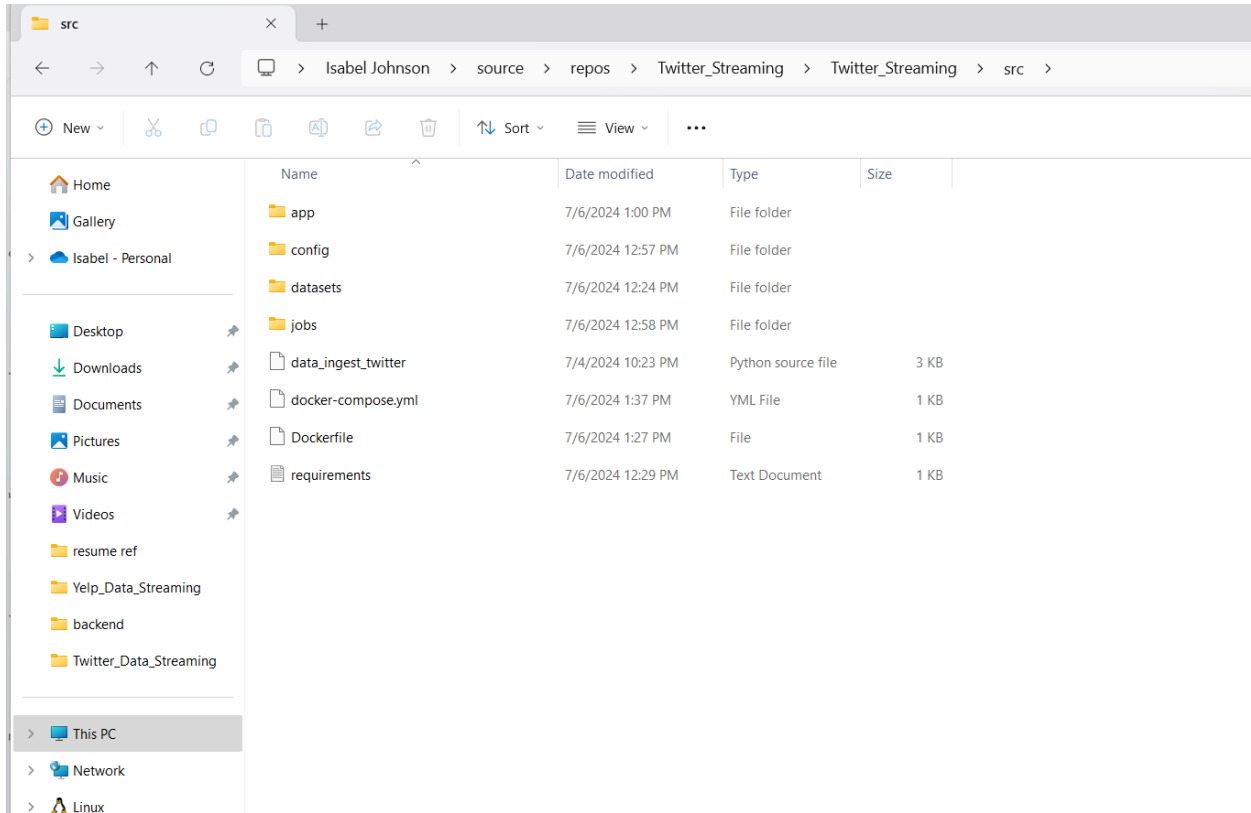
2.4.2 Steps

2.4.2.1 Docker Compose

In this design, we use `docker-compose.yml` to deploy MongoDB, React App, and Flask server. To build this setup, you need Docker software installed on your local machine.

1. Navigate to the Project Directory:

```
C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src
```



2. Run Docker Compose: Right-click on an empty space in the folder and click on "Open with Terminal". Type

In the terminal, type: `docker-compose up --build`

```

(env) C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src>docker-compose up --build
time="2024-07-06T13:39:00-04:00" level=warning msg="Found orphan containers ([flaskapp]) for this project. If you remove
d or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Running 1/0
✔ Container mongodb Created 0.0s
Attaching to mongodb
mongodb | {"t":{"$date":"2024-07-06T17:39:01.193+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"main","msg":"Au
tomatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.198+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"main","msg":"In
itialized wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":21},"incomin
gInternalClient":{"minWireVersion":0,"maxWireVersion":21},"outgoing":{"minWireVersion":6,"maxWireVersion":21},"isIntern
alClient":true}}}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.199+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main","msg":"Im
plicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQ
ueueSize."}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.205+00:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"main","msg":"Su
ccessfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","namespace":"config.tenantMigr
ationDonors"}}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.205+00:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"main","msg":"Su
ccessfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","namespace":"config.tenant
MigrationRecipients"}}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.205+00:00"},"s":"I", "c":"CONTROL", "id":5945603, "ctx":"main","msg":"Mu
lti threading initialized"}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.205+00:00"},"s":"I", "c":"TENANT_M", "id":7091600, "ctx":"main","msg":"St
arting TenantMigrationAccessBlockerRegistry"}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.207+00:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten",
"msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"aec876727e9d"}
}
mongodb | {"t":{"$date":"2024-07-06T17:39:01.207+00:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten",

```

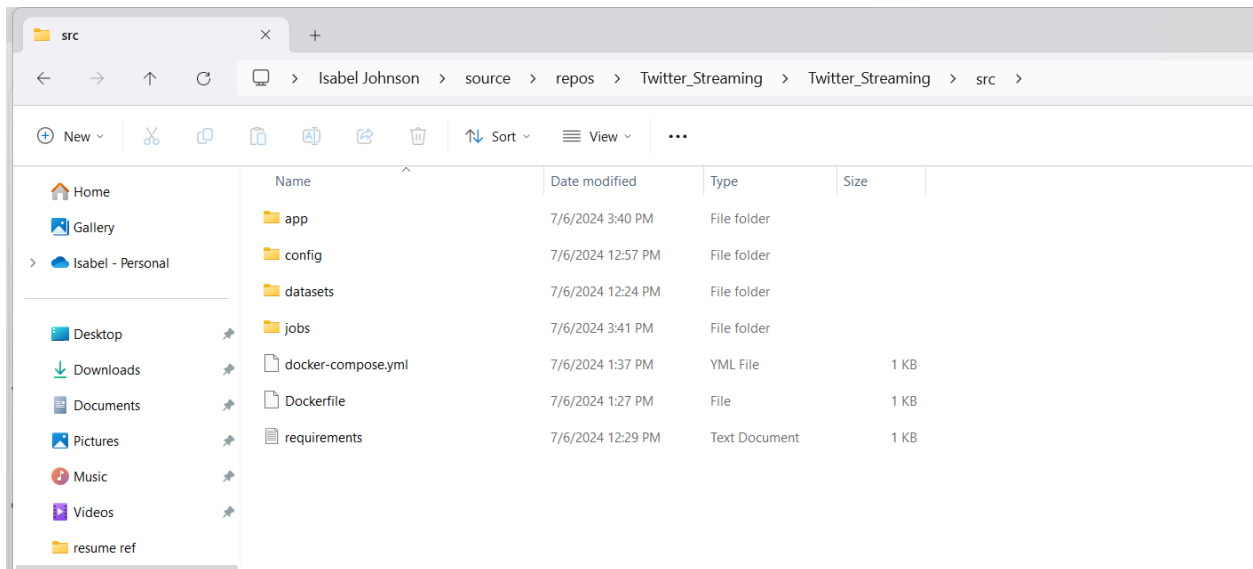
This will launch all the servers. Currently, only MongoDB is configured.

2.4.1.2 Data Loading

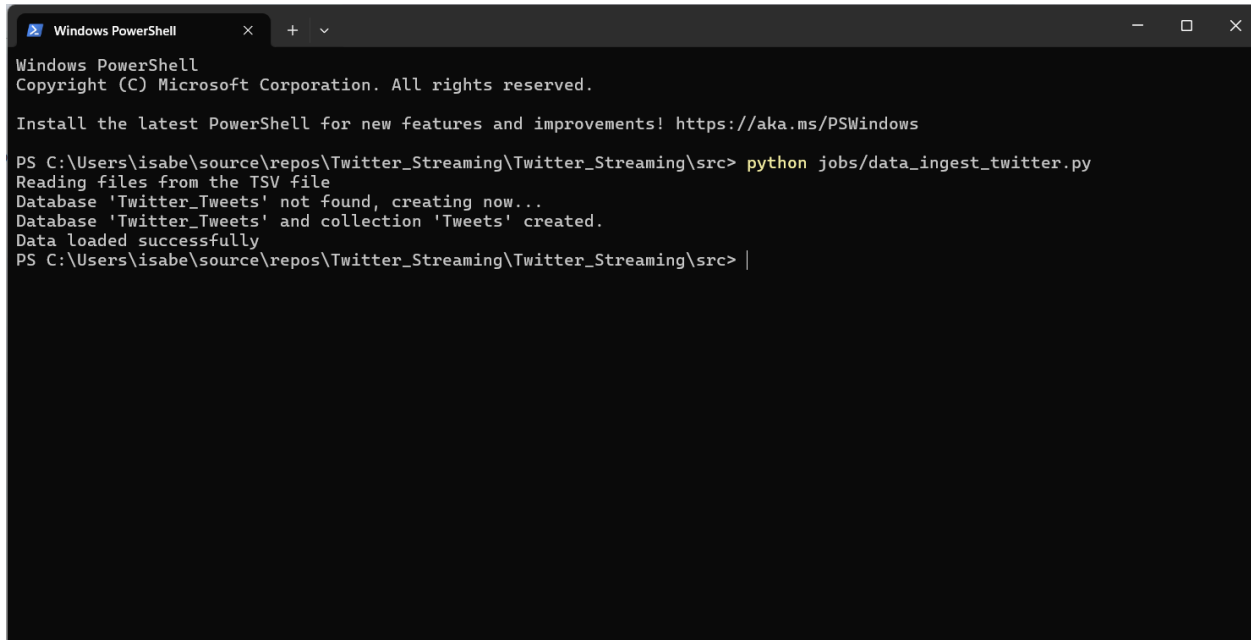
To load data into the database, follow these steps:

1. Navigate to the Jobs Folder:

`C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src\jobs`



2. Run the Data Ingestion Script: Right-click and open the terminal. Type `python jobs/data_ingest_twitter.py`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src> python jobs/data_ingest_twitter.py
Reading files from the TSV file
Database 'Twitter_Tweets' not found, creating now...
Database 'Twitter_Tweets' and collection 'Tweets' created.
Data loaded successfully
PS C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src> |
```

This will connect to MongoDB. If the `Twitter_Tweets` database does not exist, it creates the database and the `Tweets` collection, and then loads data from the dataset provided in the `datasets` folder.

Dataset: The dataset to be loaded is saved under the folder: `C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src\datasets`

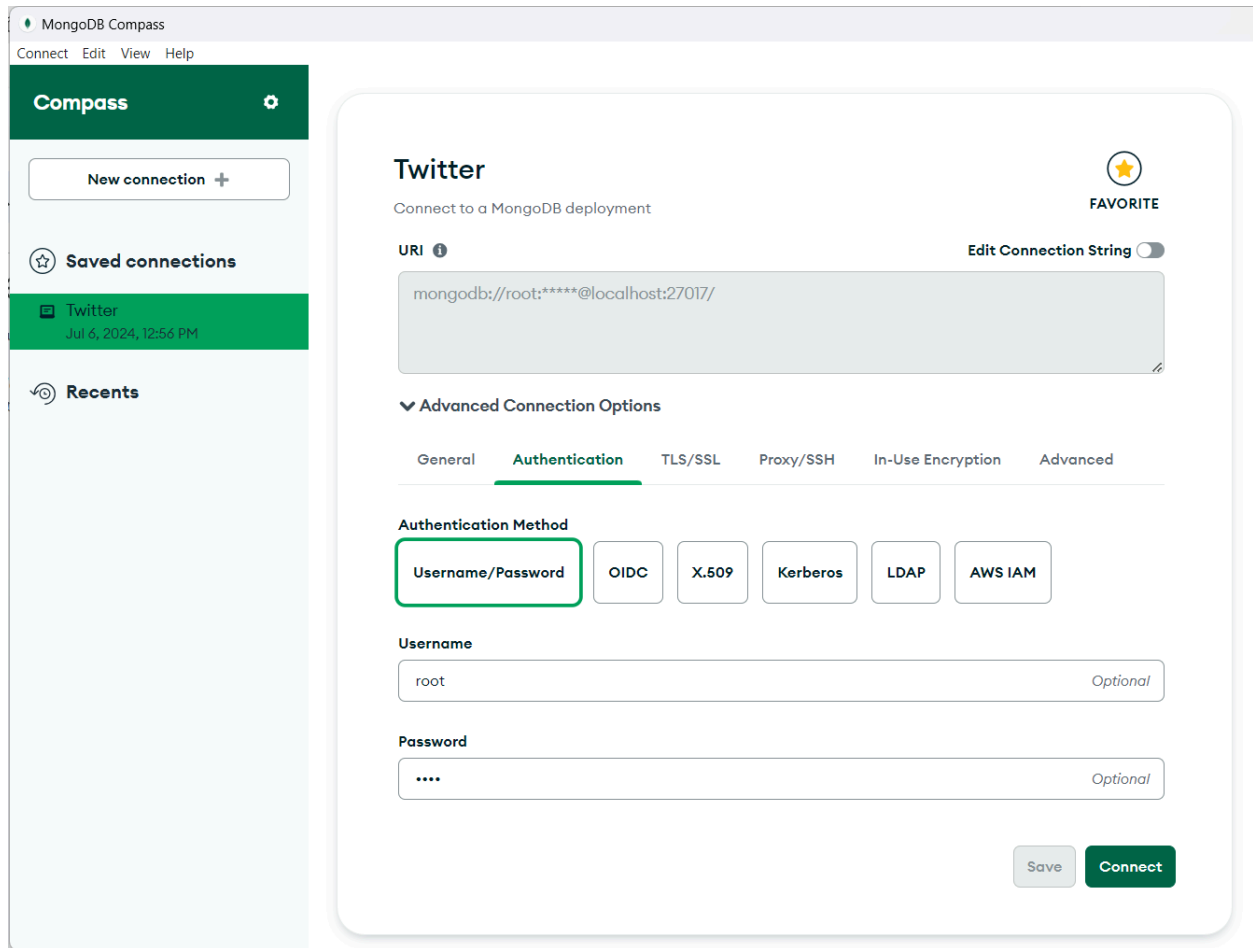
If you need to load data from a different dataset, paste the `.TSV` file under this folder.

2.4.1.3 MongoDB Connection

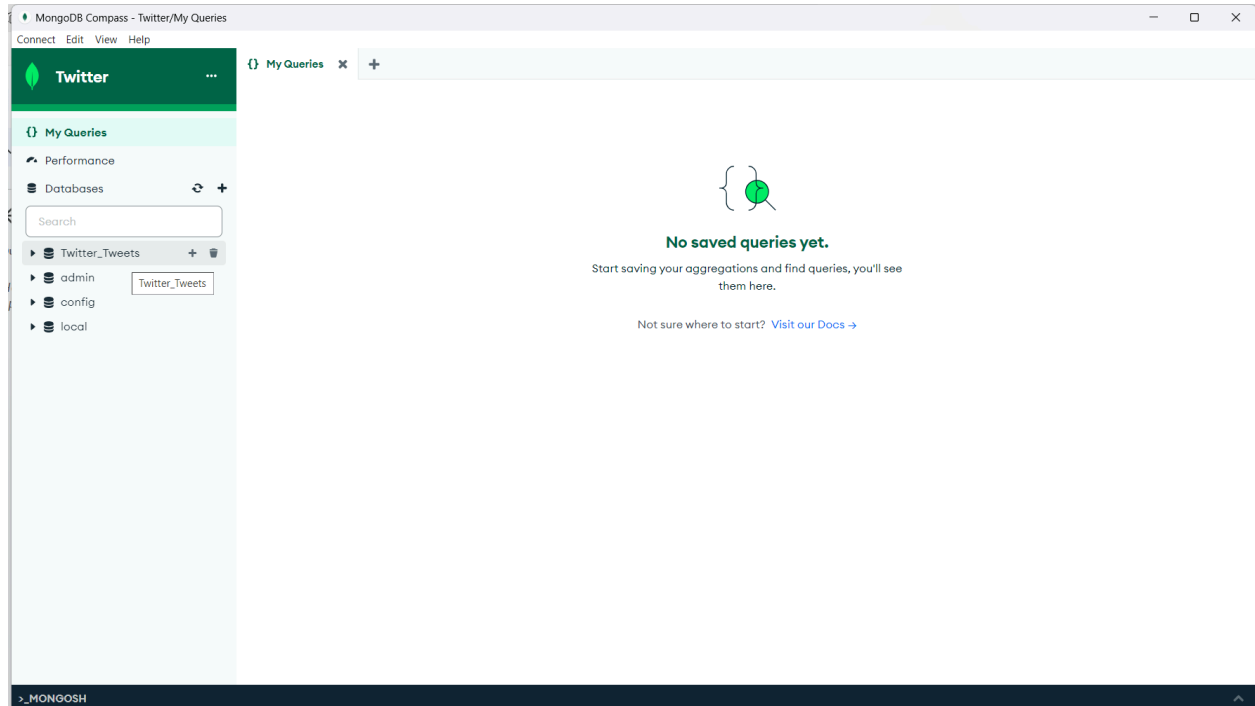
If you have the MongoDB app installed locally, connect to the local database using the URL.

- Use authentication credentials:
 - Username: `root`
 - Password: `root`

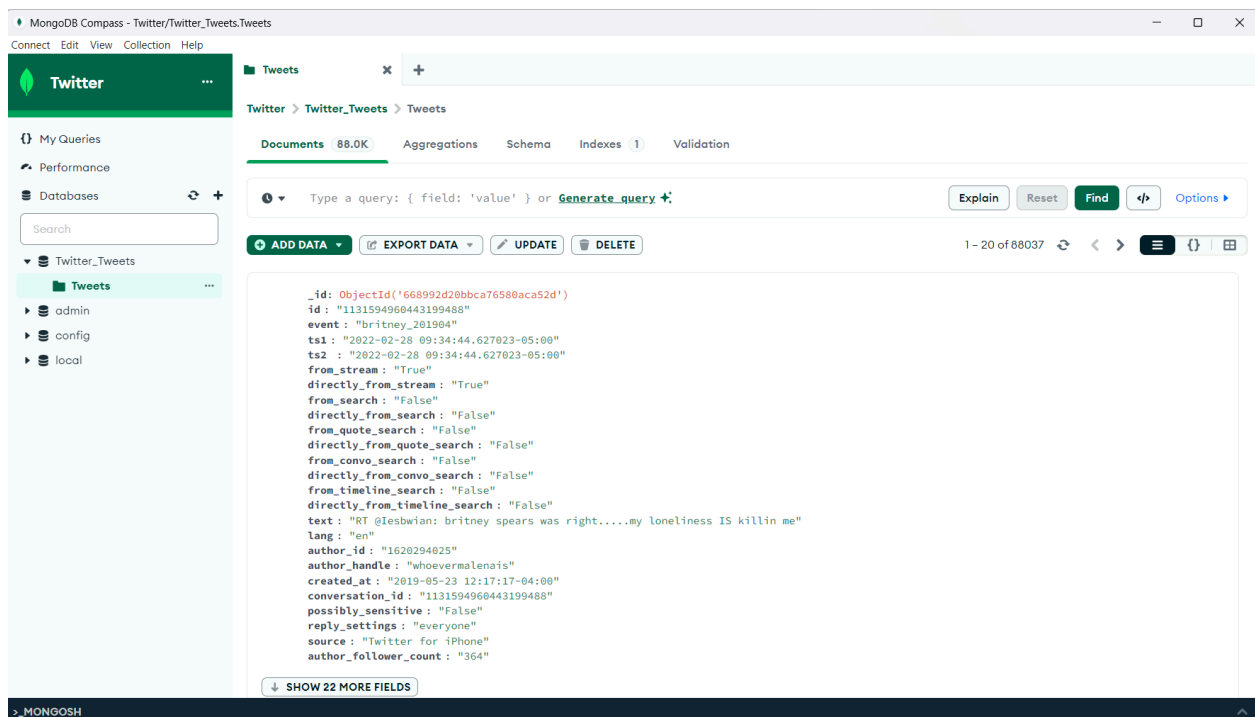
Once connected, click on `Twitter_Tweets` on the left and then on `Tweets`.



Once its connected.Click on Twitter_Tweets on the left and click on Tweets.



The results are visible like this.

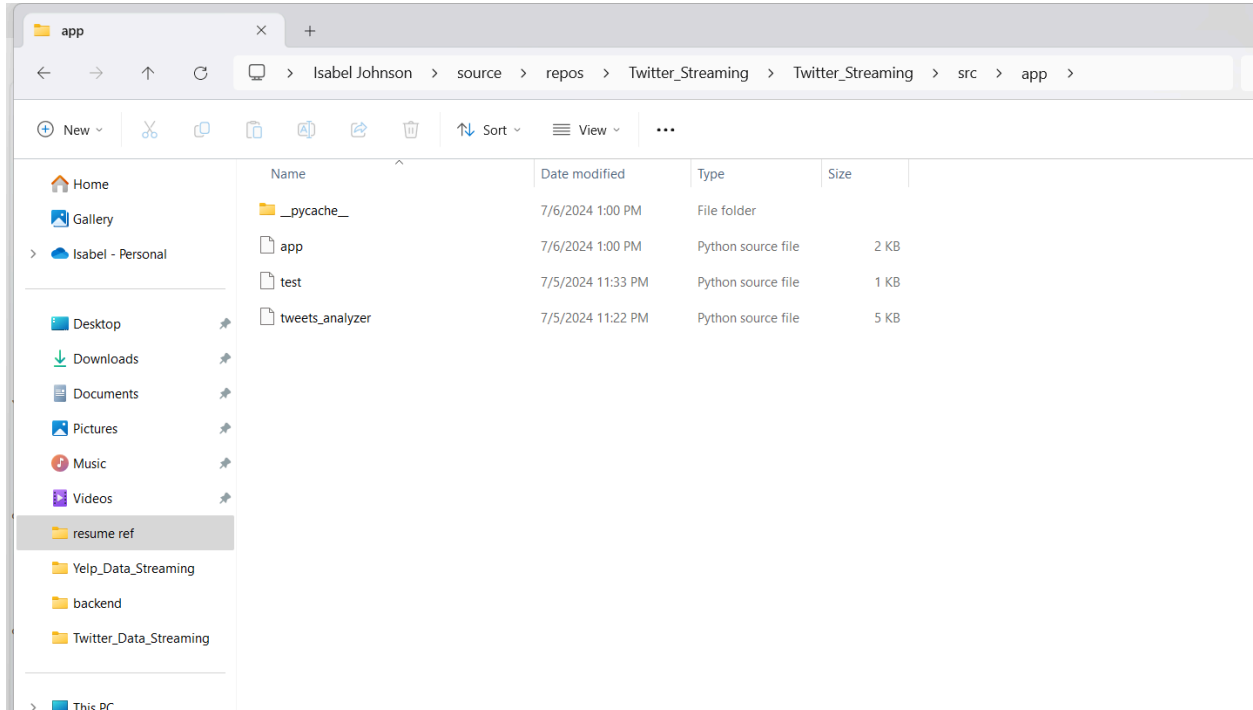


2.4.1.4 Flask Server

To start the Flask server:

1. Navigate to the App Folder:

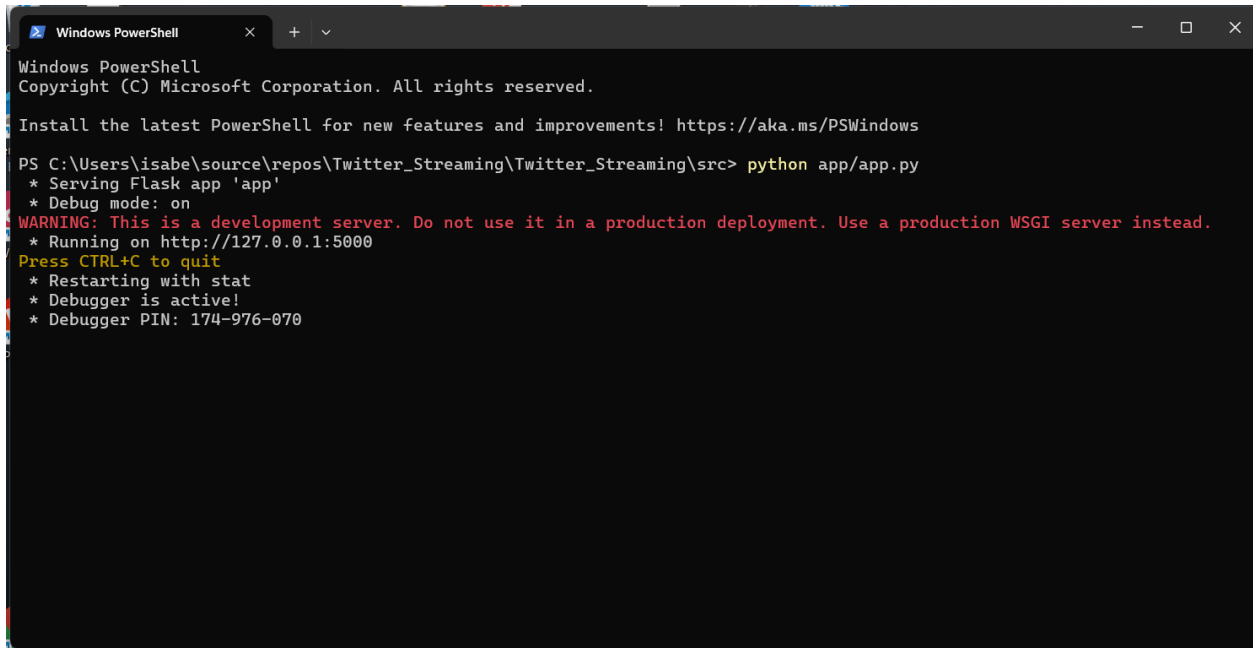
C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src\app



2. Run the Flask App: Right-click on an empty space in the folder and click on "Open with Terminal". In the terminal, type:

```
python app/app.py
```

This will launch the Flask server to send API requests to MongoDB.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

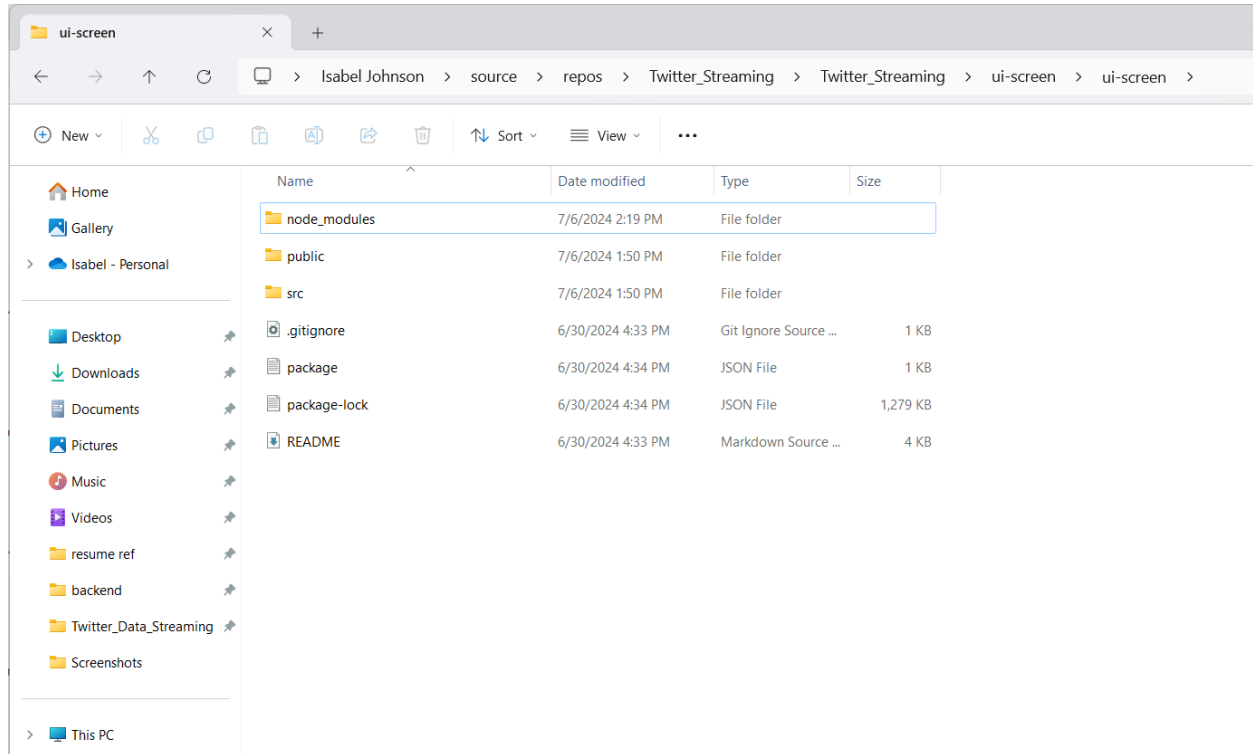
PS C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\src> python app/app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 174-976-070
```

2.4.1.5 React App

To start the React app:

1.Navigate to the UI Screen Folder:

```
C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\ui-screen\ui-screen
```



2. Run the React App: Right-click on an empty space in the folder and click on "Open with Terminal". In the terminal, type:

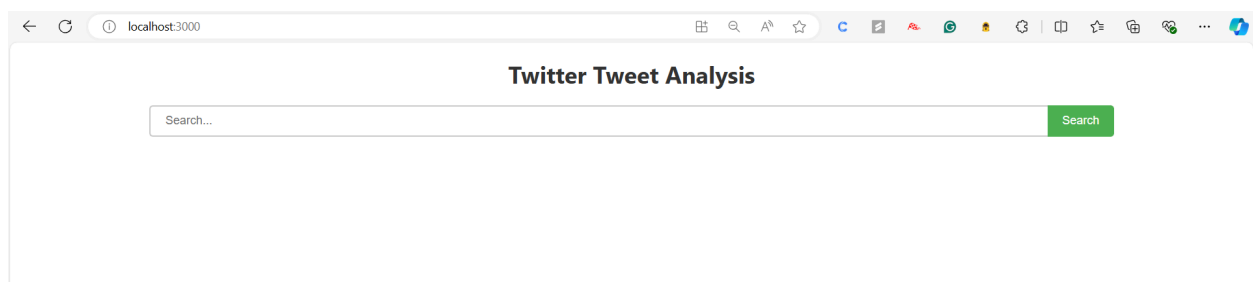
```
npm start
```

```
Windows PowerShell
PS C:\Users\isabe\source\repos\Twitter_Streaming\Twitter_Streaming\ui-screen\ui-screen> npm start

> ui-screen@0.1.0 start
> react-scripts start

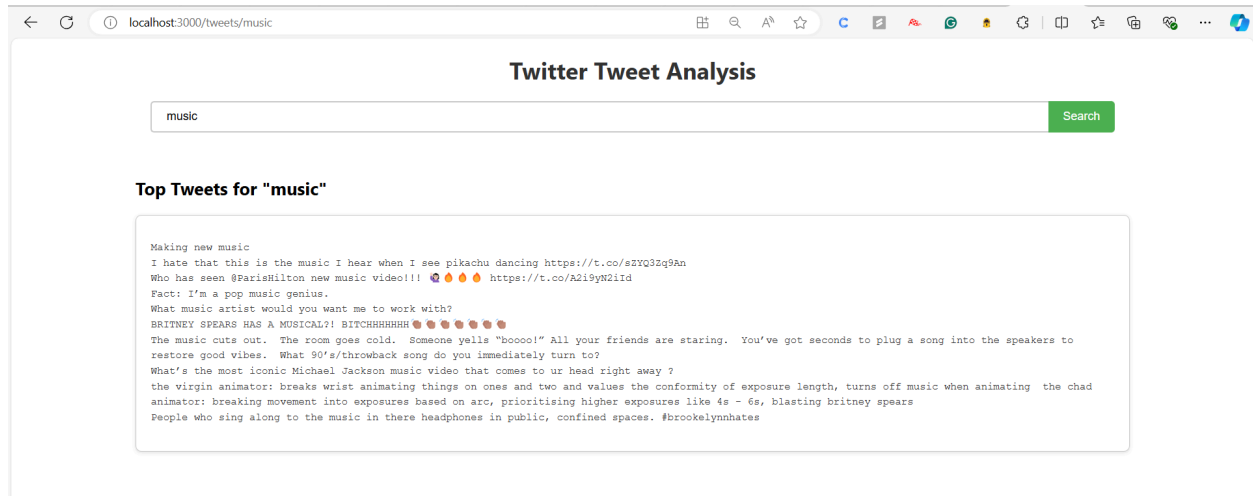
(node:26864) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:26864) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
```

This will launch the React app and open the UI screen.



2.5 Using the Application

In the UI screen, type the term you want to search (e.g., "music") and click "Search". This will fetch the top 10 tweets related to the search term.



The API request returns a JSON response, which is processed in the frontend. The JSON response includes:

```
results = {
  'top_10_tweets',
  'tweets_per_day',
  'avg_likes',
  'tweets_per_hour',
  'tweets_per_weekday',
  'tweets_per_month',
  'top_10_days',
  'bottom_10_days'
}
```

2.5.1 Viewing Full JSON Response

- Launch the Postman app.
- In a new query window, select the request as GET.

In the URL, type:

<http://127.0.0.1:5000/tweets/music>

Click "Send"

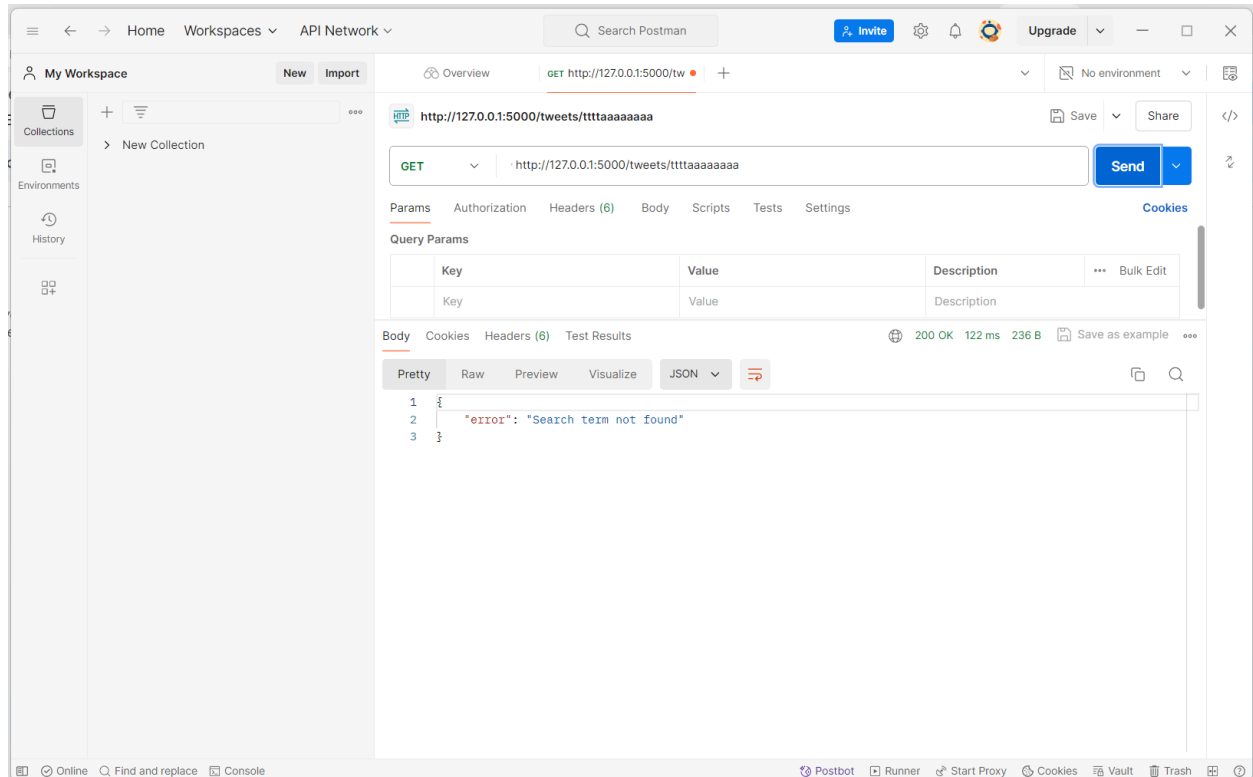
The response will display the results. You can see the status code as **200 OK**, indicating the request was successful.

The screenshot shows the Postman application interface. The URL bar displays `http://127.0.0.1:5000/tweets/music`. The request method is set to **GET**. The response status is **200 OK** with a response time of **3.30 s** and a size of **4.08 KB**. The response body is displayed in the **JSON** view, showing a list of tweets. The first tweet is:

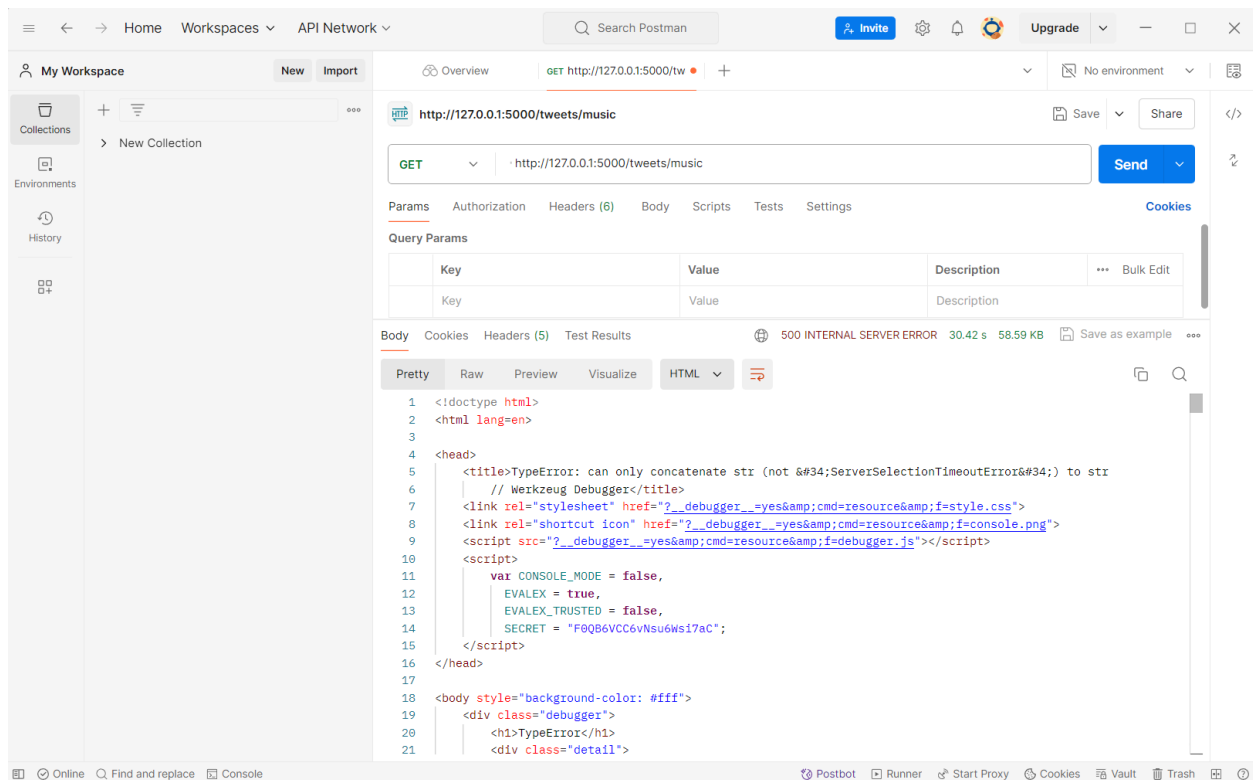
```
1 {\"top_10_tweets\": [{\"id\": \"1134155278219714560\", \"text\": \"Making new music\", \"like_count\": 288866}, {\"id\": \"1125945150830546944\", \"text\": \"I hate that this is the music I hear when I see pikachu dancing https://t.co/sZVQ3Zq9An\", \"like_count\": 74131}, {\"id\": \"1133433224837812224\", \"text\": \"who has seen [U+2866]ParisHilton[U+2869] new music video!!! \", {\"id\": \"1131725418275516416\", \"text\": \"Fact: I'm a pop music genius.\", \"like_count\": 39468}, {\"id\": \"1126820119127052292\", \"text\": \"What music artist would you want me to work with?\", \"like_count\": 18386}, {\"id\": \"110555308187918337\", \"text\": \"BRITNEY SPEARS HAS A MUSICAL?\", \"like_count\": 29. \"1131989415537127432\", \"text\": \"The music cuts out. The room goes cold. Someone yells 'boooo!' All your friends are staring. You've got seconds to plug a song into the speakers to restore good vibes. What 90's/throwback song do you immediately turn to?\", \"like_count\": 2724}, {\"id\": \"1123974634972512261\", \"text\": \"What's the most iconic Michael Jackson music video that comes to ur head right away?\", \"like_count\": 2313}, {\"id\": \"1125514223063199744\", \"text\": \"the virgin animator: breaks wrist animating things on ones and two and values the conformity of exposure length, turns off music when animating the chad animator: breaking movement into exposures based on arc, prioritising higher exposures like 4s - 6s, blasting britney spears\", \"like_count\": 1808}, {\"id\": \"112284737828538777\", \"text\": \"People who sing along to the music in there headphones in public, confined spaces. #brookelynnhates\", \"like_count\": 1782}], \"tweets_per_day\": {\"2019-03-12\": 3, \"2019-04-06\": 1, \"2019-04-14\": 1, \"2019-04-16\": 1, \"2019-04-21\": 1, \"2019-04-24\": 1, \"2019-04-26\": 2, \"2019-04-27\": 2, \"2019-04-28\": 32, \"2019-04-29\": 163, \"2019-04-30\": 93,
```

2.5.2 Handling Errors

If the search term is not found, you will receive an appropriate error message.



If MongoDB is not running, an error message will be displayed.



3. Conclusion

To conclude, the proposed architecture for this small-scale web application harnesses the combined strengths of MongoDB, Flask, React, and Docker to efficiently manage and visualize Twitter data. By leveraging MongoDB's flexibility, Flask's lightweight structure, React's dynamic interfaces, and Docker's consistent environments, the system ensures robustness, scalability, and ease of use. By developing a robust system for ingesting and querying datasets of significant size, the solution efficiently meets the defined objectives, providing insightful analytics on specific search terms. The design is well-suited for the demands of real-time data processing and user interaction, providing a streamlined approach to data ingestion, storage, and visualization. Additionally, clear guidelines and error handling measures further enhance the system's usability and reliability. This architecture not only meets the current requirements efficiently but also lays a solid foundation for future expansion and adaptation to new needs or larger datasets.