Interactive Customer Data Dashboard: A MongoDB-Powered Streamlit Application for Real-Time Data Visualization & Management.

Database Design

<u>Team Members - Group 17</u>

Bolleddu Pradeep - 02085417

1. Application Requirements / Use Cases:

1. Application Requirements / Use Cases

Objective: To develop an interactive Customer Database Dashboard using Streamlit and MongoDB. The objective is to design and implement an interactive Customer Database Dashboard using Streamlit and MongoDB. This dashboard aims to provide a user-friendly interface for managing customer records, allowing users to perform essential operations such as adding, updating, and deleting records. By leveraging Streamlit's powerful web interface capabilities and MongoDB's robust data storage, the dashboard will offer real-time data interaction and visualization. The goal is to create an efficient and intuitive tool that facilitates seamless data management, enhances user experience, and supports effective decision-making through interactive data handling and visualizations.

Features:

- **Data Display:** Showcase current data and allow users to view it in a tabular format.
- **Filtering:** Enable filtering of data by city or country to refine the view.
- **Visualization:** Generate charts to visualize customer distribution by city.

Objectives

1. ***Data Organization & Integrity:*** Properly defined tables and relationships ensure that data is organized efficiently, making it easier to store, retrieve, and manage information.
2. ***Data Normalization:*** Learning how to normalize data reduces redundancy and ensures that the database is optimized for both storage and performance for Data Engineering Problems.

Use Cases and Challenges

In this project, managing and visualizing customer data from a MongoDB database presented several challenges. One major issue was handling large volumes of data efficiently. MongoDB's flexibility with unstructured data can lead to complex queries and increased load times, especially as the dataset grows. Ensuring real-time data retrieval and updates without compromising performance required careful optimization of queries and database indexes. Additionally, visualizing extensive data sets posed its own difficulties; generating meaningful and accurate visualizations from large datasets needed effective aggregation and summarization techniques to prevent performance bottlenecks and ensure clarity in the displayed information.

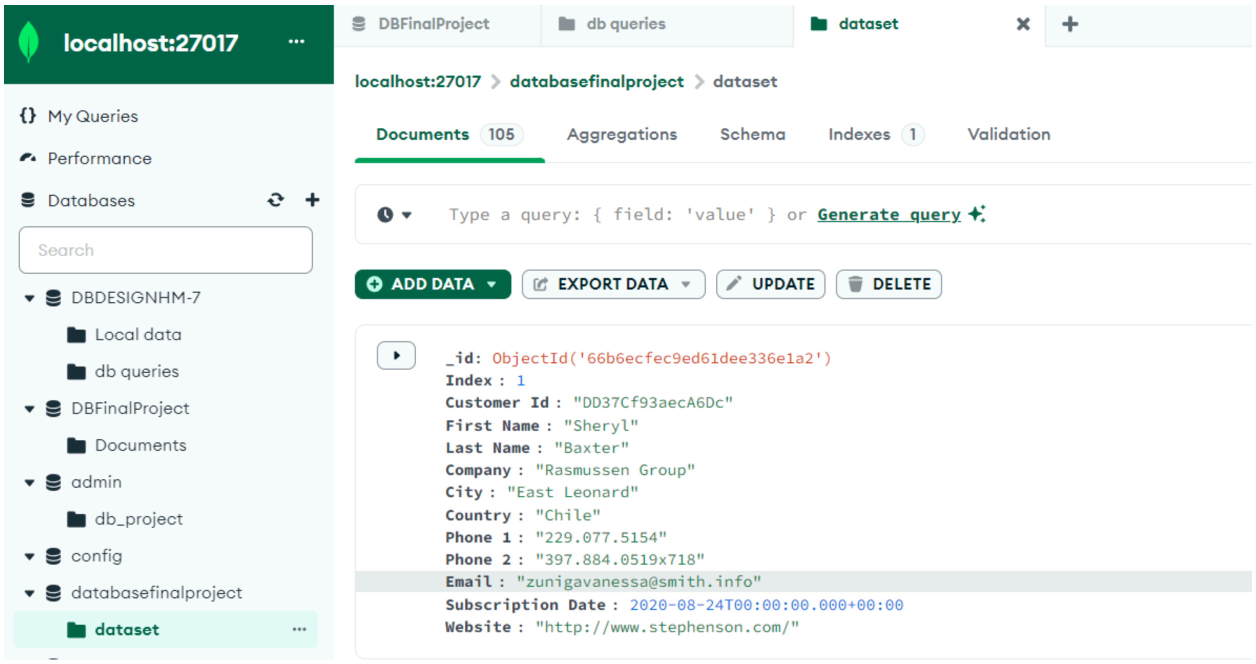
Balancing the need for a responsive user interface with the demands of processing and visualizing big data was crucial to achieving an effective and user-friendly dashboard.

- **Data Accessibility:** Difficulty in accessing and presenting data in an intuitive and interactive manner.
- **Filtering and Querying:** Need for robust filtering options to refine data based on specific criteria.
- **Visualization Needs:** Requirement to visualize data trends and distributions to make informed decisions.
- **Database Operations:** Requirement to perform Create, Read, Update, and Delete (CRUD) operations seamlessly through the user interface.

Solution: Create a comprehensive dashboard with Streamlit that integrates with MongoDB to address these issues, offering a user-friendly experience for managing and analyzing customer data.

2. Database Engine

Sample picture from the Mongo db of the imported dataset of the customer data



```
_id: ObjectId('66b6ecfec9ed61dee336e1a2')
Index : 1
Customer Id : "DD37Cf93aecA6Dc"
First Name : "Sheryl"
Last Name : "Baxter"
Company : "Rasmussen Group"
City : "East Leonard"
Country : "Chile"
Phone 1 : "229.077.5154"
Phone 2 : "397.884.0519x718"
Email : "zunigavanessa@smith.info"
Subscription Date : 2020-08-24T00:00:00.000+00:00
Website : "http://www.stephenson.com/"
```

Performed CRUD Operations

1. Create (➕ Add Record):

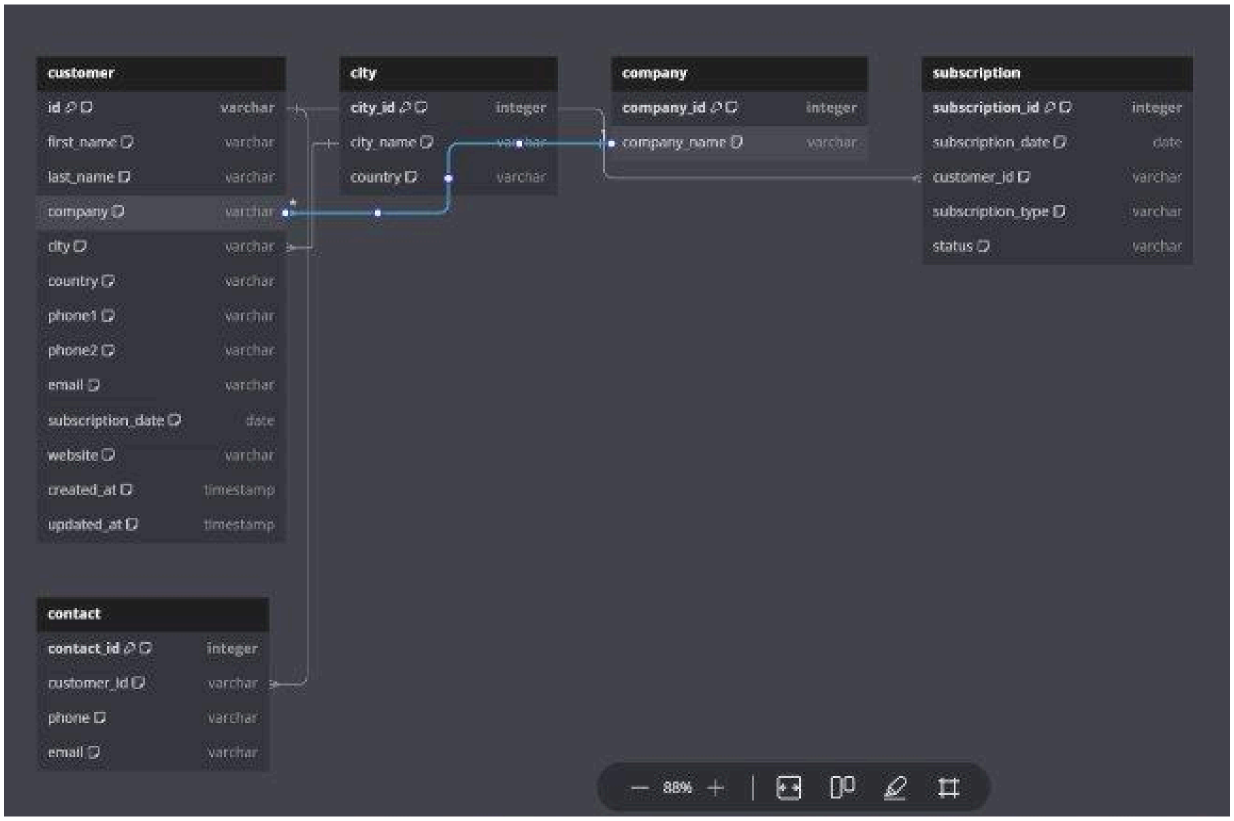
- Add new customer data (City, Country, Customers).
- Data is instantly stored in MongoDB with a confirmation.

2. Read (👁️ Show Data & 🔍 Filter):
   - View and filter customer data by City or Country.
   - Real-time data display ensures up-to-date information.

3. Update (Edit Record):
   - Modify existing records (City, Country, Customers).
   - Update by specifying the record's MongoDB `_id`.

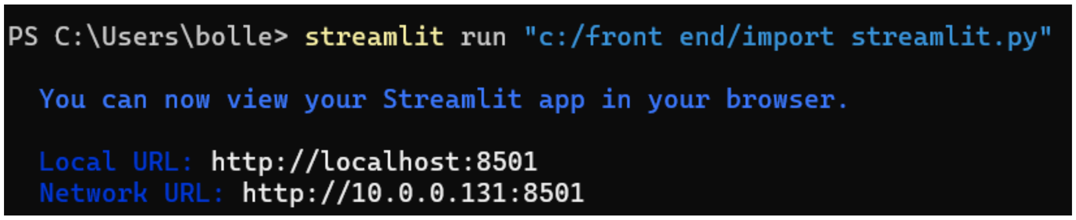
4. Delete (Remove Record):
   - Delete specific records by entering the MongoDB `_id`.
   - Confirm deletion with an on-screen success message.
   - Data Visualization: Bar chart showing customer distribution by city.

ER Diagram:
We have created ER because and The following tables are customer, city,company ,subscription and Contact ; form the customers data.



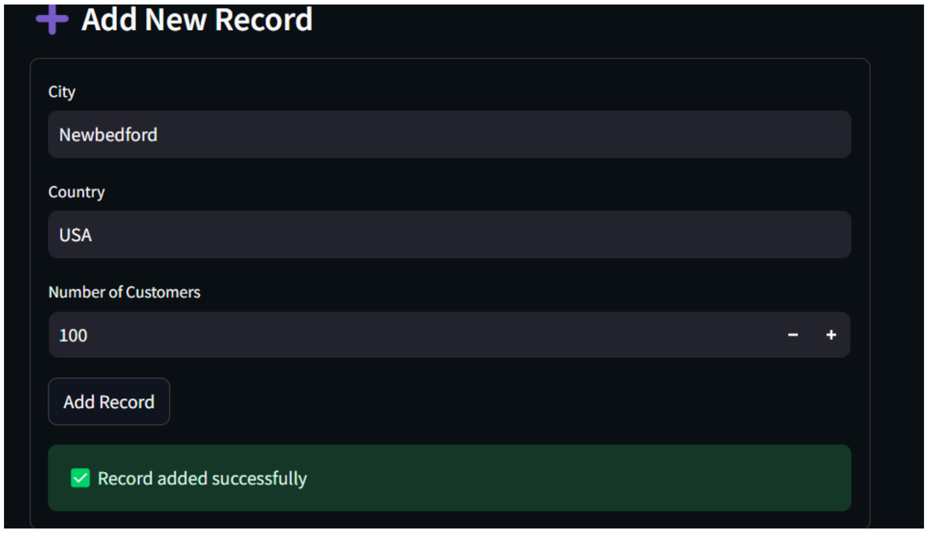
3. Front End Interface

After running the streamlit run command as shown in the image: Local rul and network url will get connected and it will display the website.



```
PS C:\Users\bolle> streamlit run "c:/front end/import streamlit.py"

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://10.0.0.131:8501
```

1. Here we added the New record to the database from the created front end:
   We utilized ***Streamlit*** to create a user-friendly front-end interface, connecting it to MongoDB as the backend database. This setup enabled us to add a new record to the database smoothly. The integration of Streamlit for the UI and MongoDB for data storage allowed for a streamlined process, making it easy for users to interact with the system and manage data efficiently.



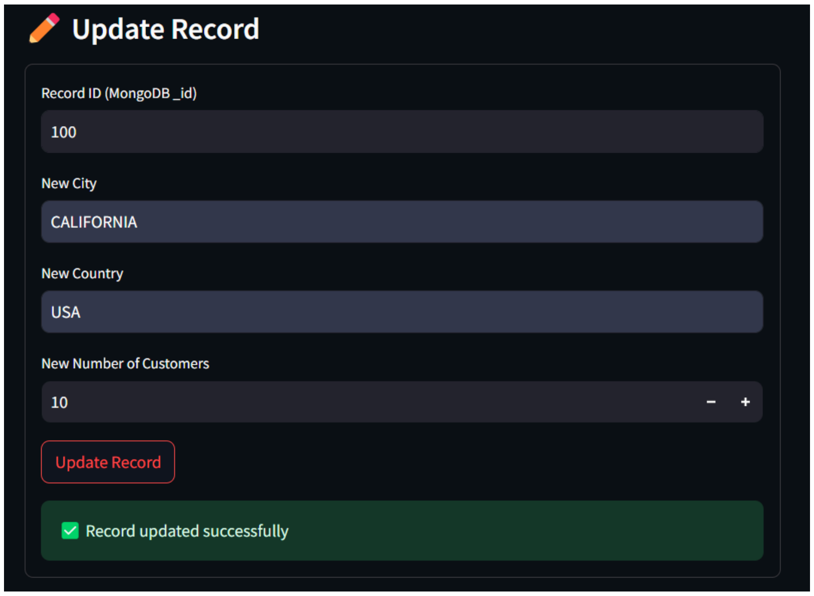
Add New Record

City	Newbedford
Country	USA
Number of Customers	100

✓ Record added successfully

2. ***Here we updated the record from the frontend page, as shown in the image :***

Using Streamlit, we developed an intuitive front-end interface that seamlessly connects with a MongoDB backend. This configuration allowed us to update an existing record in the database efficiently. The combination of Streamlit's user-friendly design and MongoDB's robust data management made the process smooth, enabling users to interact with the application and modify data effortlessly.



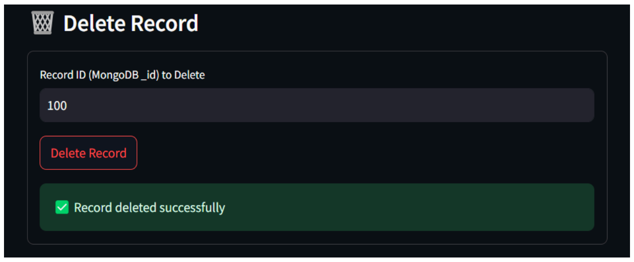
Update Record

Record ID (MongoDB _id)	100
New City	CALIFORNIA
New Country	USA
New Number of Customers	10

✓ Record updated successfully

3. **Deleting the record using the front interface**
   We designed a user-friendly interface with Streamlit that allows for record deletion directly from the front end. This setup effectively communicates with MongoDB, enabling

users to delete records from the database with ease and ensuring a smooth interaction between the interface and the backend.


Delete Record

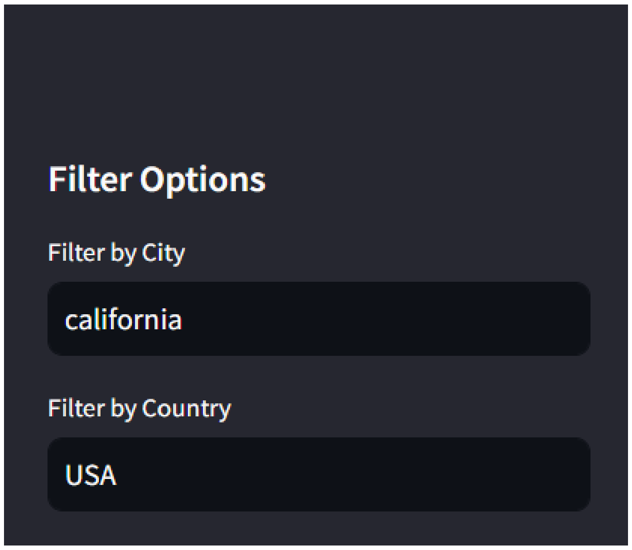
Record ID (MongoDB _id) to Delete

100

✓ Record deleted successfully

4. Filtering out the city and the country

We developed a filtering feature in Streamlit to refine data based on city and country. This functionality was designed to enhance user experience by allowing precise searches within the database. Users can input specific city and country criteria on the front end, which communicates with MongoDB to fetch and display only the relevant records. This setup improves data management efficiency and helps users quickly locate specific information. By leveraging Streamlit's interactive capabilities and MongoDB's data handling, we ensured a smooth and responsive filtering process.


Filter Options

Filter by City

california

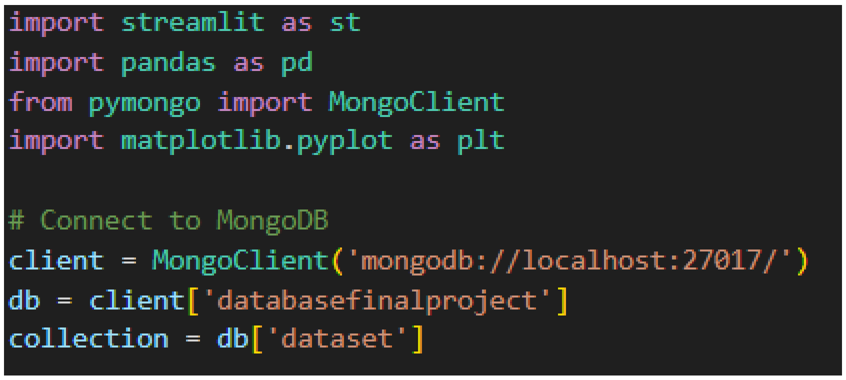
Filter by Country

USA

4. Source Code

- ***Displaying and Querying the data from the local connected Database***

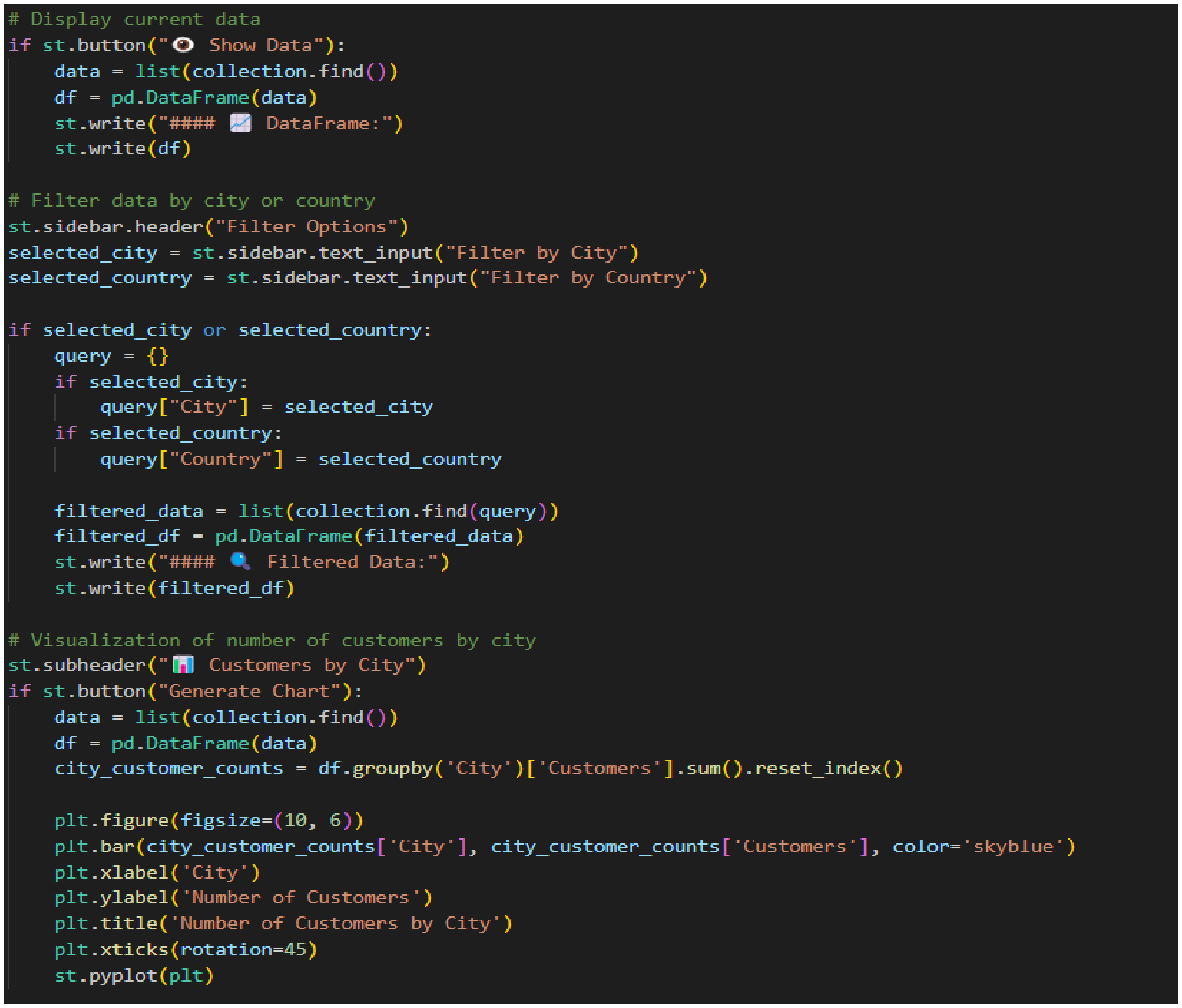
This code snippet sets up a ***Streamlit*** application with connections to MongoDB. It begins by importing necessary libraries: Streamlit for the app interface, pandas for data manipulation, pymongo for interacting with MongoDB, and matplotlib for plotting. Next, it establishes a connection to a local MongoDB instance, selects the database finalproject database, and accesses the dataset collection within it. This setup allows the app to query and work with data stored in MongoDB.



```python
import streamlit as st
import pandas as pd
from pymongo import MongoClient
import matplotlib.pyplot as plt

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['databasefinalproject']
collection = db['dataset']
```

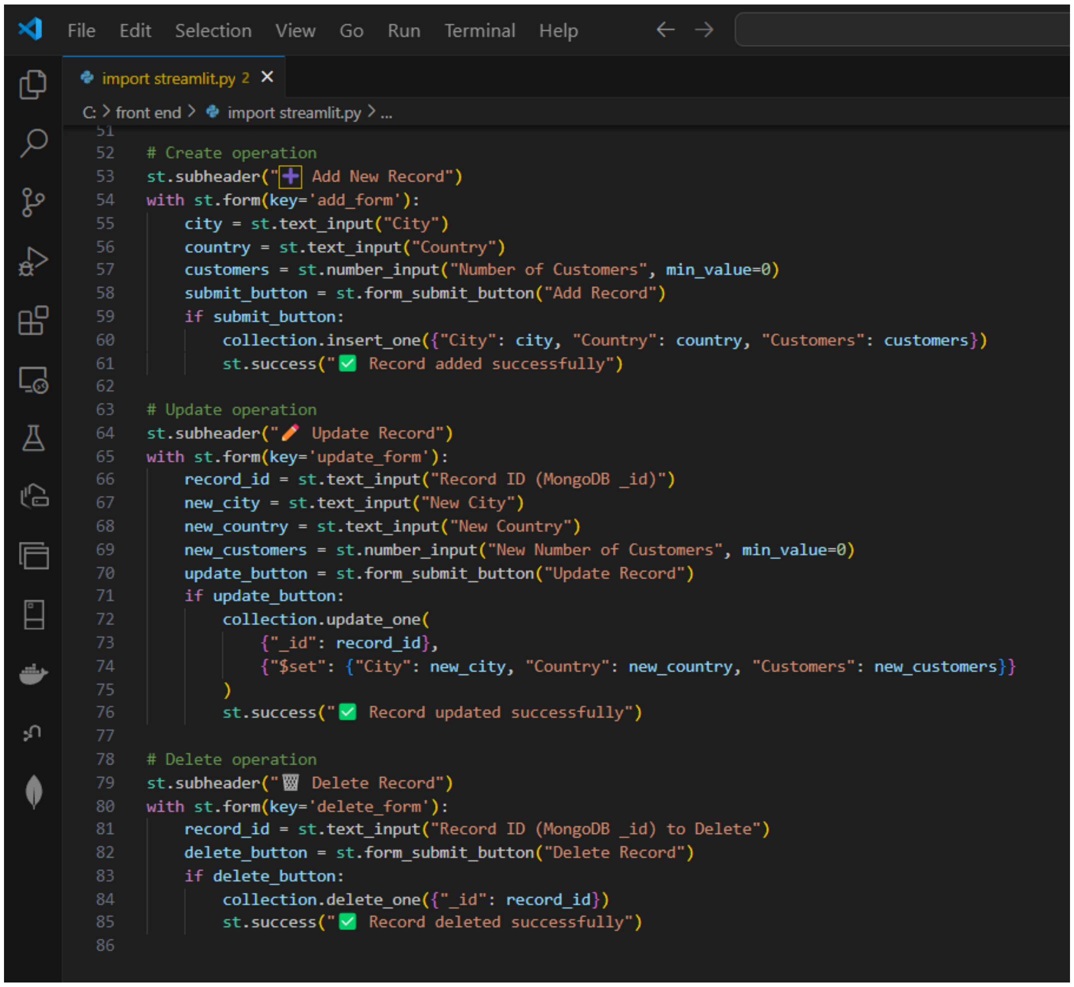
- **Filtering and Querying the Data**

  This Streamlit app provides a user-friendly interface for interacting with a customer database. The app begins by setting a title for the dashboard and allows users to view all current data from the MongoDB collection when the "***Show Data***" button is pressed. Users can filter this data by ***city*** or ***country*** using the sidebar inputs, which updates the displayed data based on their criteria. Additionally, the app features a visualization section where users can generate a bar chart showing the number of customers by city. This chart is created by aggregating the data from MongoDB and is displayed using ***matplotlib***.



```python
# Display current data
if st.button("👁 Show Data"):
    data = list(collection.find())
    df = pd.DataFrame(data)
    st.write("#### 📈 DataFrame:")
    st.write(df)

# Filter data by city or country
st.sidebar.header("Filter Options")
selected_city = st.sidebar.text_input("Filter by City")
selected_country = st.sidebar.text_input("Filter by Country")

if selected_city or selected_country:
    query = {}
    if selected_city:
        query["City"] = selected_city
    if selected_country:
        query["Country"] = selected_country

    filtered_data = list(collection.find(query))
    filtered_df = pd.DataFrame(filtered_data)
    st.write("#### 🔍 Filtered Data:")
    st.write(filtered_df)

# Visualization of number of customers by city
st.subheader("📊 Customers by City")
if st.button("Generate Chart"):
    data = list(collection.find())
    df = pd.DataFrame(data)
    city_customer_counts = df.groupby('City')['Customers'].sum().reset_index()

    plt.figure(figsize=(10, 6))
    plt.bar(city_customer_counts['City'], city_customer_counts['Customers'], color='skyblue')
    plt.xlabel('City')
    plt.ylabel('Number of Customers')
    plt.title('Number of Customers by City')
    plt.xticks(rotation=45)
    st.pyplot(plt)
```

- **Creating the CRUD Operations :**
  This Streamlit app includes features for managing records in a customer database. Users can add new records by filling out a form with city, country, and number of customers, and then submit it to insert the data into MongoDB. For updating existing records, users enter the MongoDB record ID along with the new details for city, country, and number of customers, which the app uses to modify the corresponding entry in the database. Additionally, users can delete records by specifying the record ID; the app will remove the record from MongoDB and confirm the deletion. **Each operation—creation, update, and deletion—includes user feedback to ensure successful execution.**



```python
# Create operation
st.subheader("➕ Add New Record")
with st.form(key='add_form'):
    city = st.text_input("City")
    country = st.text_input("Country")
    customers = st.number_input("Number of Customers", min_value=0)
    submit_button = st.form_submit_button("Add Record")
    if submit_button:
        collection.insert_one({"City": city, "Country": country, "Customers": customers})
        st.success("✅ Record added successfully")

# Update operation
st.subheader("✏️ Update Record")
with st.form(key='update_form'):
    record_id = st.text_input("Record ID (MongoDB _id)")
    new_city = st.text_input("New City")
    new_country = st.text_input("New Country")
    new_customers = st.number_input("New Number of Customers", min_value=0)
    update_button = st.form_submit_button("Update Record")
    if update_button:
        collection.update_one(
            {"_id": record_id},
            {"$set": {"City": new_city, "Country": new_country, "Customers": new_customers}}
        )
        st.success("✅ Record updated successfully")

# Delete operation
st.subheader("🗑 Delete Record")
with st.form(key='delete_form'):
    record_id = st.text_input("Record ID (MongoDB _id) to Delete")
    delete_button = st.form_submit_button("Delete Record")
    if delete_button:
        collection.delete_one({"_id": record_id})
        st.success("✅ Record deleted successfully")

```

Conclusion

We created a structured database schema by defining individual tables for various customer attributes and establishing relationships between them. This approach improved data organization, integrity, and query efficiency. By understanding these principles, we've gained essential skills for designing scalable and maintainable databases, crucial for data-driven applications.