**Lab 10 -Connecting a Python Application to a MySQL Database**

In this lab, you will learn how to connect a python application on streamlit with a MySQL database. The entire procedure is described in the following steps:

**Objectives:**

1. Create a CRUD Application using Streamlit
2. Connect the application to MySQL Server

**Step 1: Installations**

**Tools required:**

1. **Python:** <https://www.python.org/downloads/>
2. **PyCharm:** PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers. <https://www.jetbrains.com/pycharm/download/#section=windows>
3. **Streamlit:** It is an open source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries. Command: *pip install streamlit*
4. **MySQL:** It is a widely used relational database management system (RDBMS). <https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/>
5. **MySQL WorkBench**: MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. <https://dev.mysql.com/downloads/workbench/>

Note: MySQL Workbench is a MySQL Server GUI. It requires a MySQL Server connection for most tasks. Documentation: [WorkBench](https://www.mysql.com/products/workbench/)

**Step 2: Connecting the application to a MySQL Server**

Once a new project is created in PyCharm, a file named ‘app.py’ is created inside the project. When the code is executed for the first time, the below snippet is executed to create a database named ‘ebike’. Upon execution, the code is commented out.

**app.py:**

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="password"

)

c = mydb.cursor()

c.execute("CREATE DATABASE ebike")

**app.py:** The main function which can be used to start the application.

**MySQL Connector:** It enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API.

**mysql.connector.connect:** This method sets up a connection, establishing a session with the MySQL server. If no arguments are given, it uses the already configured or default values.

**mydb.cursor():** It is used to execute statements to communicate with the MySQL database.

Note: This code is later written into the file ‘database.py’ to connect the application to the database, a connection for the same can be created in the MySQL WorkBench.

**Step 3: Writing SQL Statements for CRUD Operations**

Creating a table inside ebike database:

CREATE TABLE IF NOT EXISTS DEALER(dealer\_id TEXT, dealer\_name TEXT, dealer\_city TEXT, dealer\_pin TEXT, dealer\_street TEXT)

Adding a dealer into the table:

INSERT INTO DEALER(dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street) VALUES (%s,%s,%s,%s,%s)

Viewing all the data that has been added:

SELECT \* FROM DEALER

Viewing only the dealers:

SELECT dealer\_name FROM DEALER

Getting a particular dealer:

SELECT \* FROM DEALER WHERE dealer\_name=“{}"

Editing an already created dealer:

UPDATE DEALER SET dealer\_id=%s, dealer\_name=%s, dealer\_city=%s, dealer\_pin=%s, dealer\_street=%s WHERE dealer\_id=%s and dealer\_name=%s and dealer\_city=%s and dealer\_pin=%s and dealer\_street=%s

Deleting a dealer:

DELETE FROM DEALER WHERE dealer\_name="{}"

**Step 4: Creating the Python Application for CRUD Operations**

Here, we will try to execute an eBike Dealer Application where we can create, view, update and delete the dealers. Code is briefly explained.

Link to code: [eBike Application](https://github.com/sirigowda204/eBike-MySQL-DBMS-Lab)

Command to run the application: *streamlit run <path of the file>*

**4.A:** The **app.py** file acts as the main function that calls other functions like **create()**, **read()**, **update()** and **delete()** which have been written as separate files for clarity.

**4.B:** Code for create.py, read.py, update.py and delete.py

* **create.py:** Here, a dealer is created and added to the database from the UI. Streamlit provides various user friendly functions like columns, selectbox etc which can be used to create an interactive UI. The records of the dealer can be visualised in MySQL WorkBench.

**app.py:**

import streamlit as st

from create import create

from database import create\_table

from delete import delete

from read import read

from update import update

def main():

st.title("eBike App")

menu = ["Add", "View", "Edit", "Remove"]

choice = st.sidebar.selectbox("Menu", menu)

create\_table()

if choice == "Add":

st.subheader("Enter Dealer Details:")

create()

elif choice == "View":

st.subheader("View created tasks")

read()

elif choice == "Edit":

st.subheader("Update created tasks")

update()

elif choice == "Remove":

st.subheader("Delete created tasks")

delete()

else:

st.subheader("About tasks")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**create.py:**

import streamlit as st

from database import add\_data

def create():

col1, col2 = st.columns(2)

with col1:

dealer\_id = st.text\_input("ID:")

dealer\_name = st.text\_input("Name:")

with col2:

dealer\_city = st.selectbox("City", ["Bangalore", "Chennai", "Mumbai"])

dealer\_pin = st.text\_input("Pin Code:")

dealer\_street = st.text\_input("Street Name:")

if st.button("Add Dealer"):

add\_data(dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street)

st.success("Successfully added dealer: {}".format(dealer\_name))

* **read.py:** In this file, you can view all the dealers added in the UI and also use plotly to visualise the locations of the dealers added in the form of a pie chart.

**read.py:**

import pandas as pd

import streamlit as st

import plotly.express as px

from database import view\_all\_data

def read():

result = view\_all\_data()

# st.write(result)

df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin', 'Dealer Street'])

with st.expander("View all Dealers"):

st.dataframe(df)

with st.expander("Dealer Location"):

task\_df = df['Dealer City'].value\_counts().to\_frame()

task\_df = task\_df.reset\_index()

st.dataframe(task\_df)

p1 = px.pie(task\_df, names='index', values='Dealer City')

st.plotly\_chart(p1)

**update.py:**

import pandas as pd

import streamlit as st

from database import view\_all\_data, view\_only\_dealer\_names, get\_dealer, edit\_dealer\_data

def update():

result = view\_all\_data()

df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin', 'Dealer Street'])

with st.expander("Current Dealers"):

st.dataframe(df)

list\_of\_dealers = [i[0] for i in view\_only\_dealer\_names()]

selected\_dealer = st.selectbox("Dealer to Edit", list\_of\_dealers)

selected\_result = get\_dealer(selected\_dealer)

if selected\_result:

dealer\_id = selected\_result[0][0]

dealer\_name = selected\_result[0][1]

dealer\_city = selected\_result[0][2]

dealer\_pin = selected\_result[0][3]

dealer\_street = selected\_result[0][4]

col1, col2 = st.columns(2)

with col1:

new\_dealer\_id = st.text\_input("ID:", dealer\_id)

new\_dealer\_name = st.text\_input("Name:", dealer\_name)

with col2:

new\_dealer\_city = st.selectbox(dealer\_city, ["Bangalore", "Chennai", "Mumbai"])

new\_dealer\_pin = st.text\_input("Pin Code:", dealer\_pin)

new\_dealer\_street = st.text\_input("Street Name:", dealer\_street)

if st.button("Update Dealer"):

edit\_dealer\_data(new\_dealer\_id, new\_dealer\_name, new\_dealer\_city, new\_dealer\_pin, new\_dealer\_street, dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street)

st.success("Successfully updated:: {} to ::{}".format(dealer\_name, new\_dealer\_name))

result2 = view\_all\_data()

df2 = pd.DataFrame(result2, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin', 'Dealer Street’])

with st.expander("Updated data"):

st.dataframe(df2)

* **update.py:** In this file, you can update the details of a selected dealer that already exists in the database and see the changes in the UI itself.
* **delete.py:** Deleting the record of a selected dealer that already exists in the database. Upon deleting a dealer in the UI, the same can be seen in the SQL Database from MySQL WorkBench.

**database.py:**

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="password",

database="ebike"

)

c = mydb.cursor()

def create\_table():

c.execute('CREATE TABLE IF NOT EXISTS DEALER(dealer\_id TEXT, dealer\_name TEXT, dealer\_city TEXT, dealer\_pin TEXT, dealer\_street TEXT)')

def add\_data(dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street):

c.execute('INSERT INTO DEALER(dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street) VALUES (%s,%s,%s,%s,%s)',(dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street))

mydb.commit()

def view\_all\_data():

c.execute('SELECT \* FROM DEALER')

data = c.fetchall()

return data

def view\_only\_dealer\_names():

c.execute('SELECT dealer\_name FROM DEALER')

data = c.fetchall()

return data

def get\_dealer(dealer\_name):

c.execute('SELECT \* FROM DEALER WHERE dealer\_name="{}"'.format(dealer\_name))

data = c.fetchall()

return data

def edit\_dealer\_data(new\_dealer\_id, new\_dealer\_name, new\_dealer\_city, new\_dealer\_pin, new\_dealer\_street, dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street):

c.execute("UPDATE DEALER SET dealer\_id=%s, dealer\_name=%s, dealer\_city=%s, dealer\_pin=%s, dealer\_street=%s WHERE dealer\_id=%s and dealer\_name=%s and dealer\_city=%s and dealer\_pin=%s and dealer\_street=%s", (new\_dealer\_id, new\_dealer\_name, new\_dealer\_city, new\_dealer\_pin, new\_dealer\_street, dealer\_id, dealer\_name, dealer\_city, dealer\_pin, dealer\_street))

mydb.commit()

data = c.fetchall()

return data

def delete\_data(dealer\_name):

c.execute('DELETE FROM DEALER WHERE dealer\_name="{}"'.format(dealer\_name))

mydb.commit()

**delete.py:**

import pandas as pd

import streamlit as st

from database import view\_all\_data, view\_only\_dealer\_names, delete\_data

def delete():

result = view\_all\_data()

df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin', 'Dealer Street'])

with st.expander("Current data"):

st.dataframe(df)

list\_of\_dealers = [i[0] for i in view\_only\_dealer\_names()]

selected\_dealer = st.selectbox("Task to Delete", list\_of\_dealers)

st.warning("Do you want to delete ::{}".format(selected\_dealer))

if st.button("Delete Dealer"):

delete\_data(selected\_dealer)

st.success("Dealer has been deleted successfully")

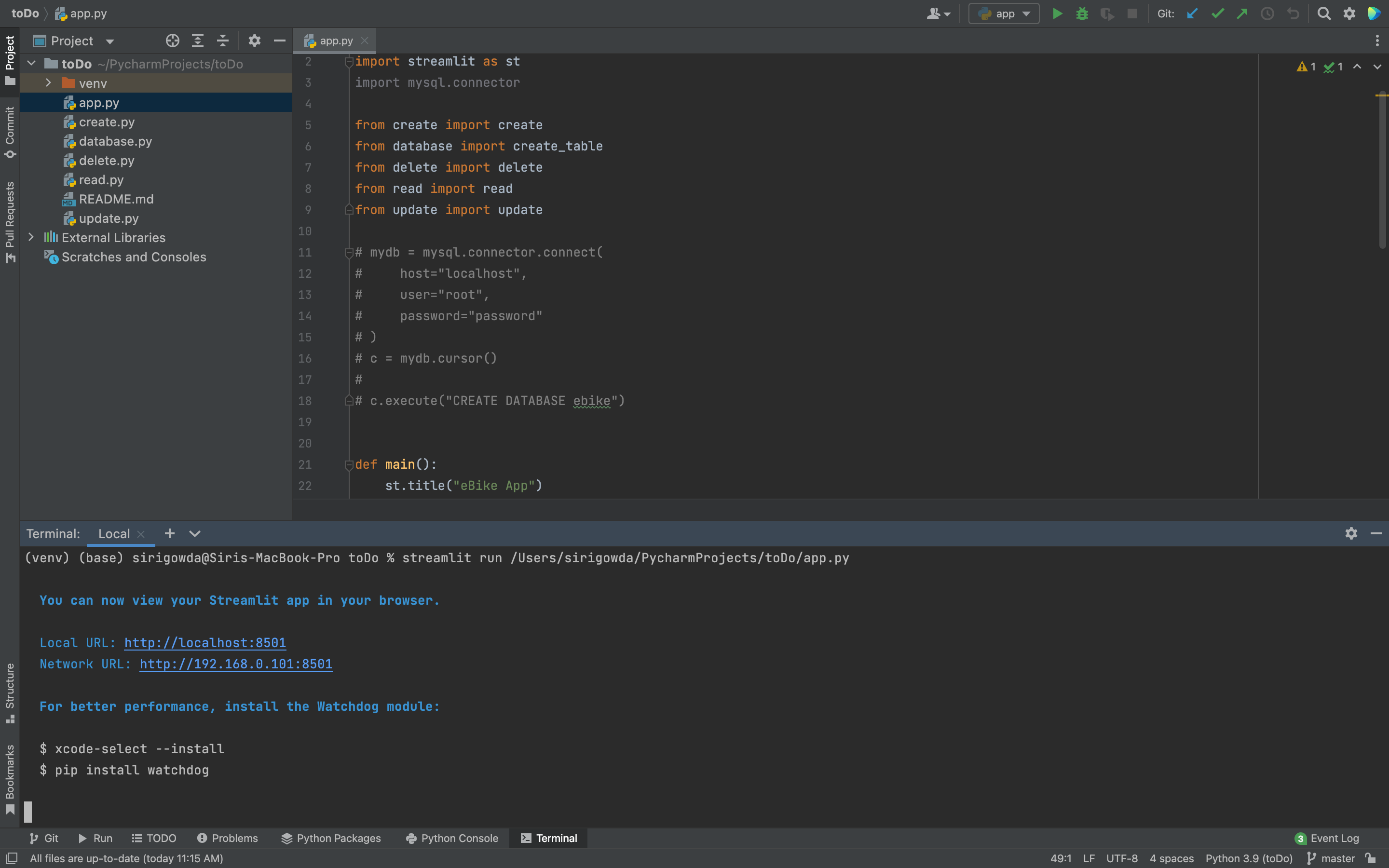
new\_result = view\_all\_data()

df2 = pd.DataFrame(new\_result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin', 'Dealer Street'])

with st.expander("Updated data"):

st.dataframe(df2)

***4.C:* database.py:** In this file, you make a connection between the server and the python application. The above SQL statements are implemented within this file.

**Running the Application:**

**Assignment: Railway Reservation**

1. Execute a CRUD (Create, Read, Update and Delete) application in python using Streamlit and MySQL to create a table ‘train’ in the User-Interface.

The ‘train’ table should be populated with the following 3 records using the User-Interface:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Train\_No** | **Name** | **Train\_Type** | **Source** | **Destination** | **Availability** |
| 62621 | BEN-CHE Shatabdi | Superfast | Bengaluru | Chennai | yes |
| 62620 | CHE-BEN Shatabdi | Fast | Chennai | Bengaluru | No |
| 25261 | Managaluru Mail | Mail | Chennai | Mangaluru | Yes |

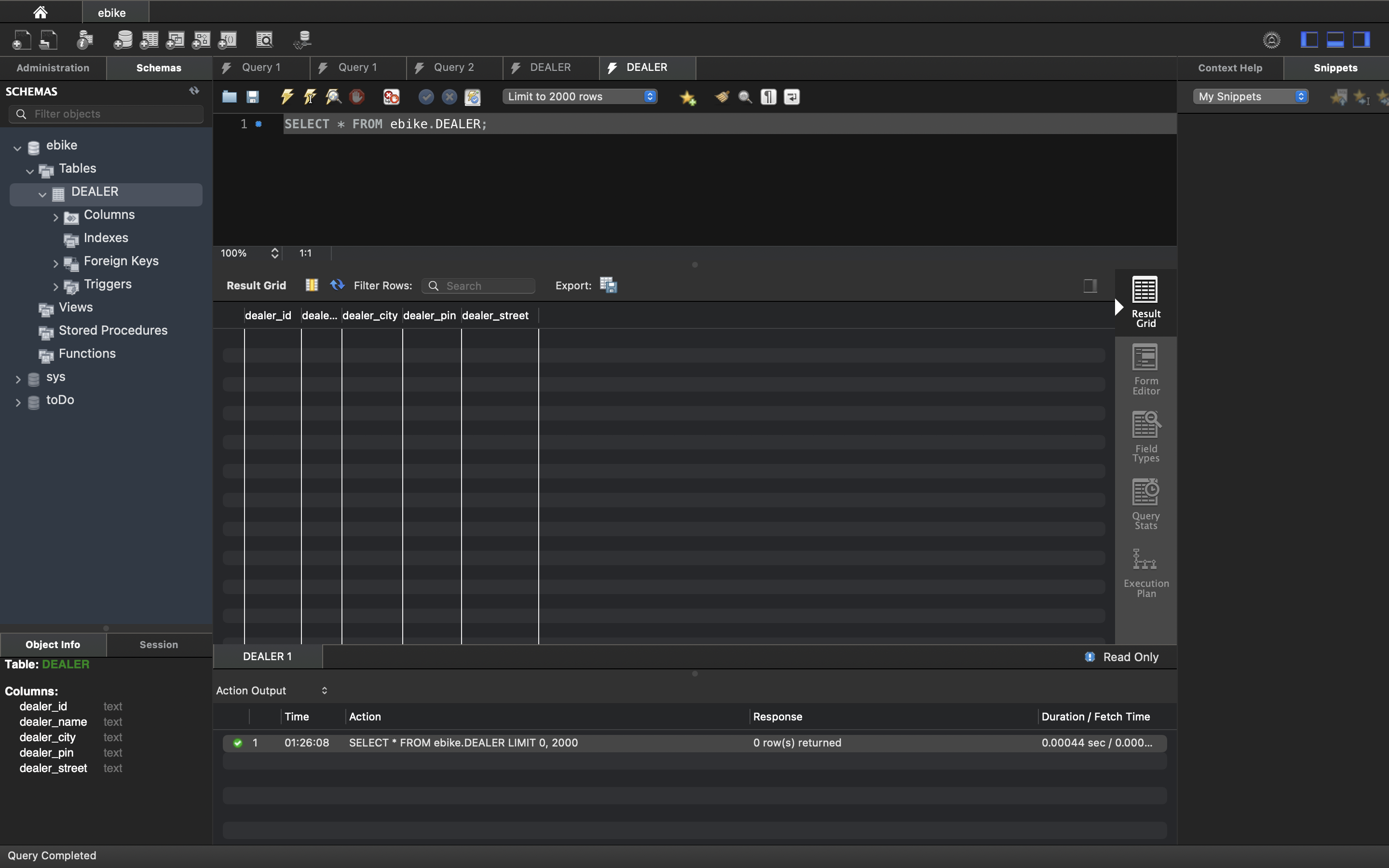
2. Read the details entered at real time in the User-Interface itself.

3. Update the ‘Availability’ of the Train\_No 62620 to ‘yes’ in the User-Interface.

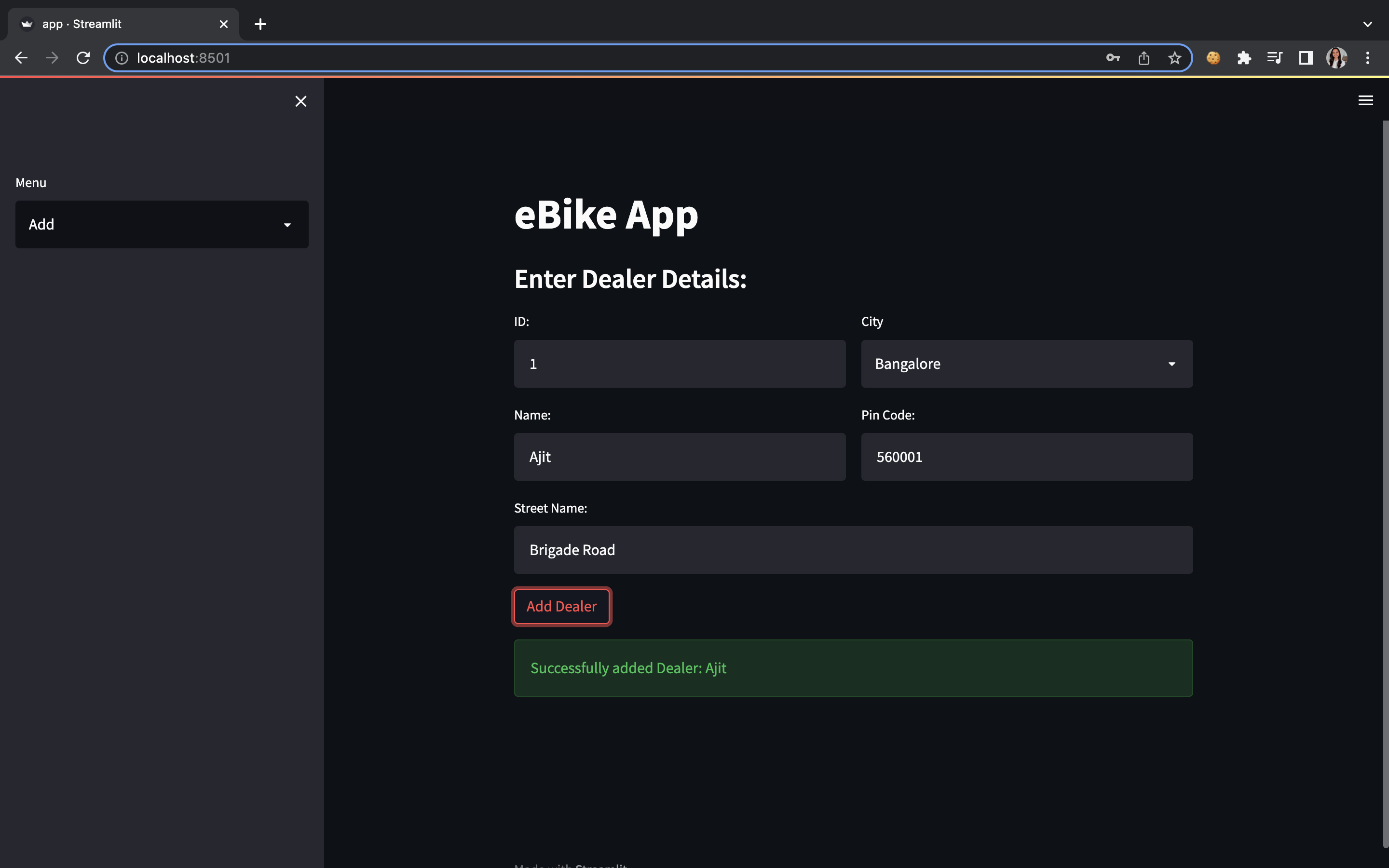
4. Delete the Train\_No 25261 in the User-Interface.

**Deliverables for submission:** Upload a PDF with the following 6 Screenshots. (Remember to incorporate your SRN)

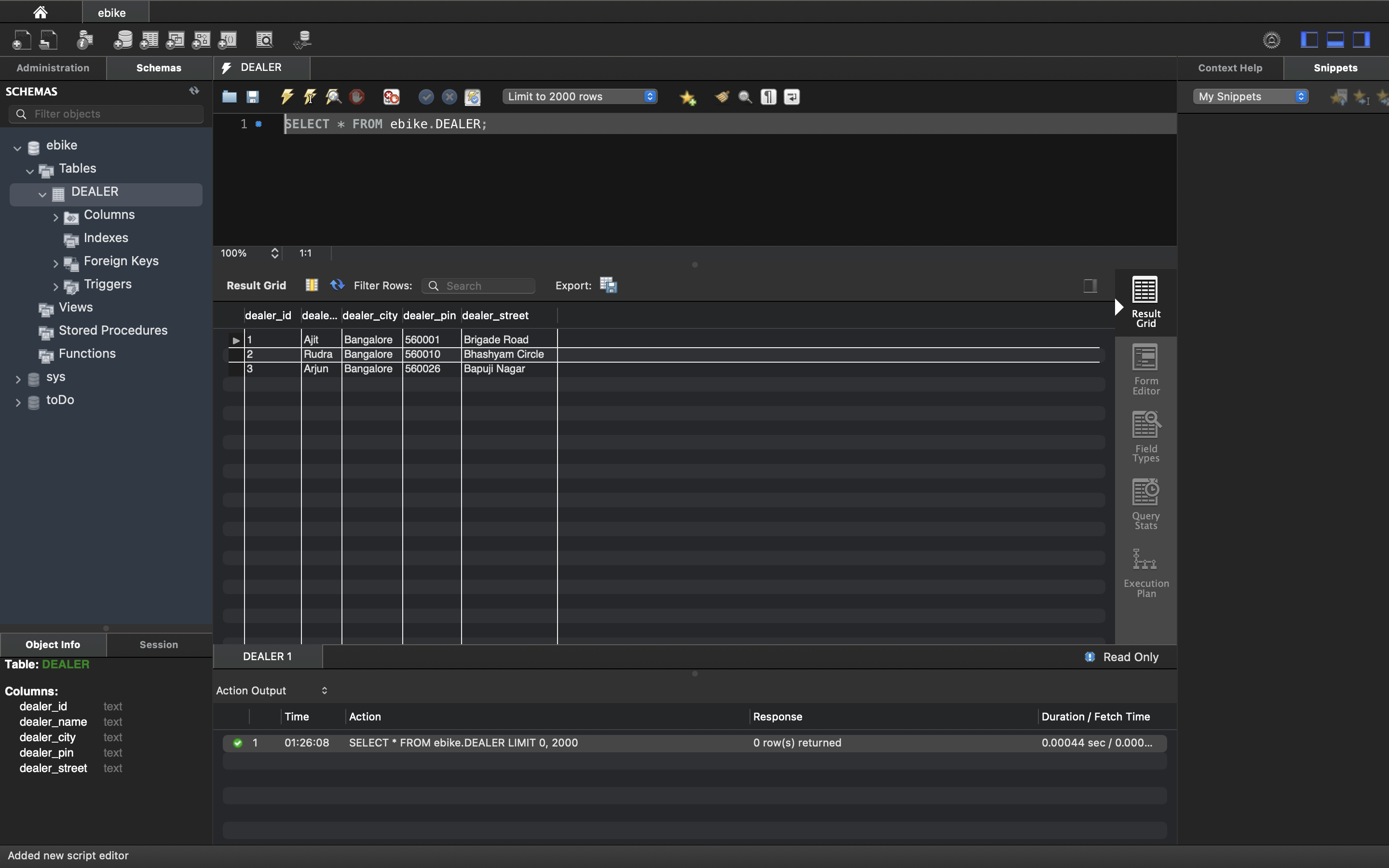
1. Screenshot of database with the table - ‘train’ before populating it.
2. Screenshot of the User Interface.
3. Screenshot of the 3 records in the train table from MySQL WorkBench.
4. Screenshot of the same 3 records visualised in the User Interface.
5. Screenshot of Updated Train\_No 62620 in the User-Interface.
6. Screenshot of User-Interface after the Train\_No 25261 has been deleted.

**Sample screenshots as per demo exercise:**

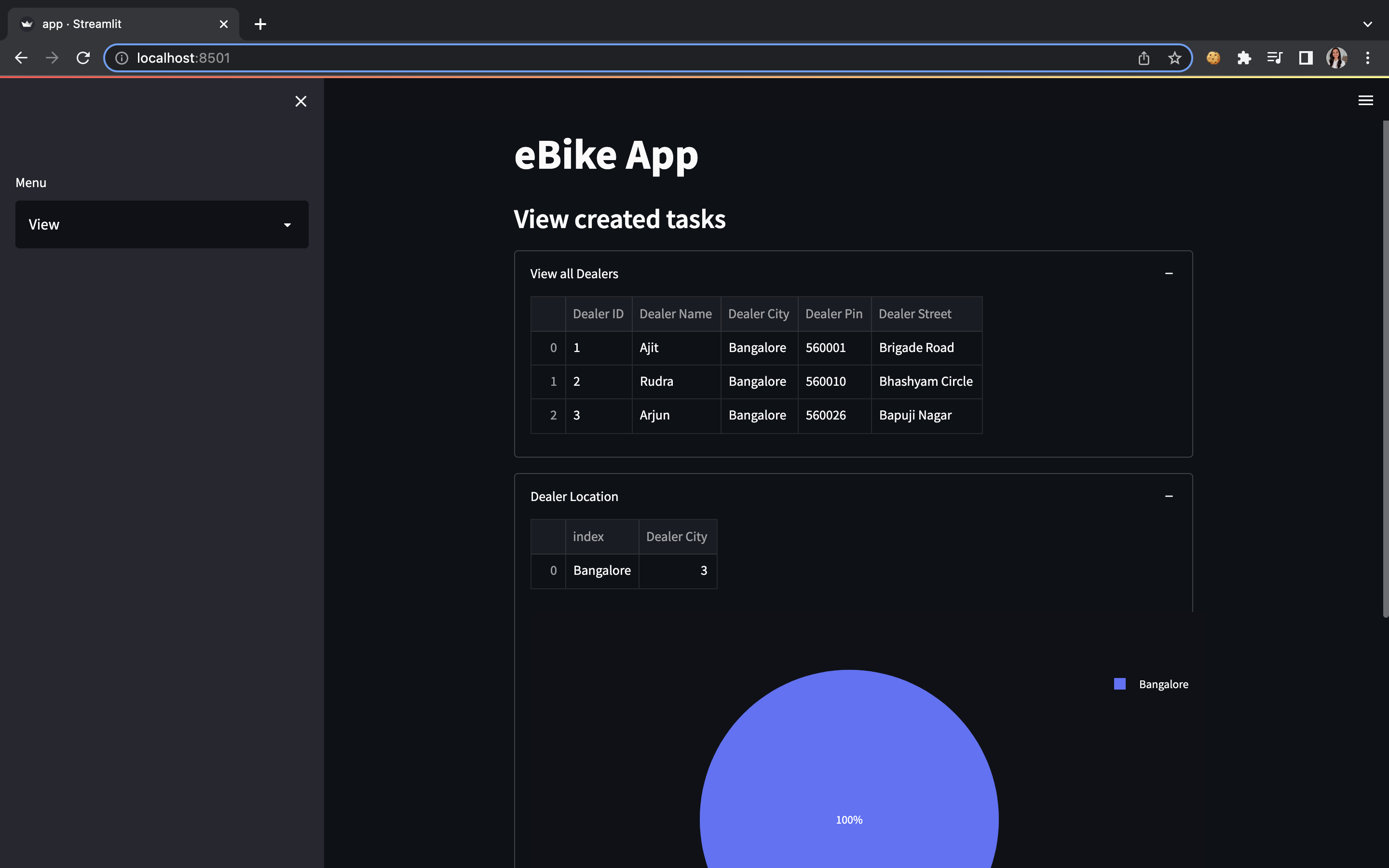
1.



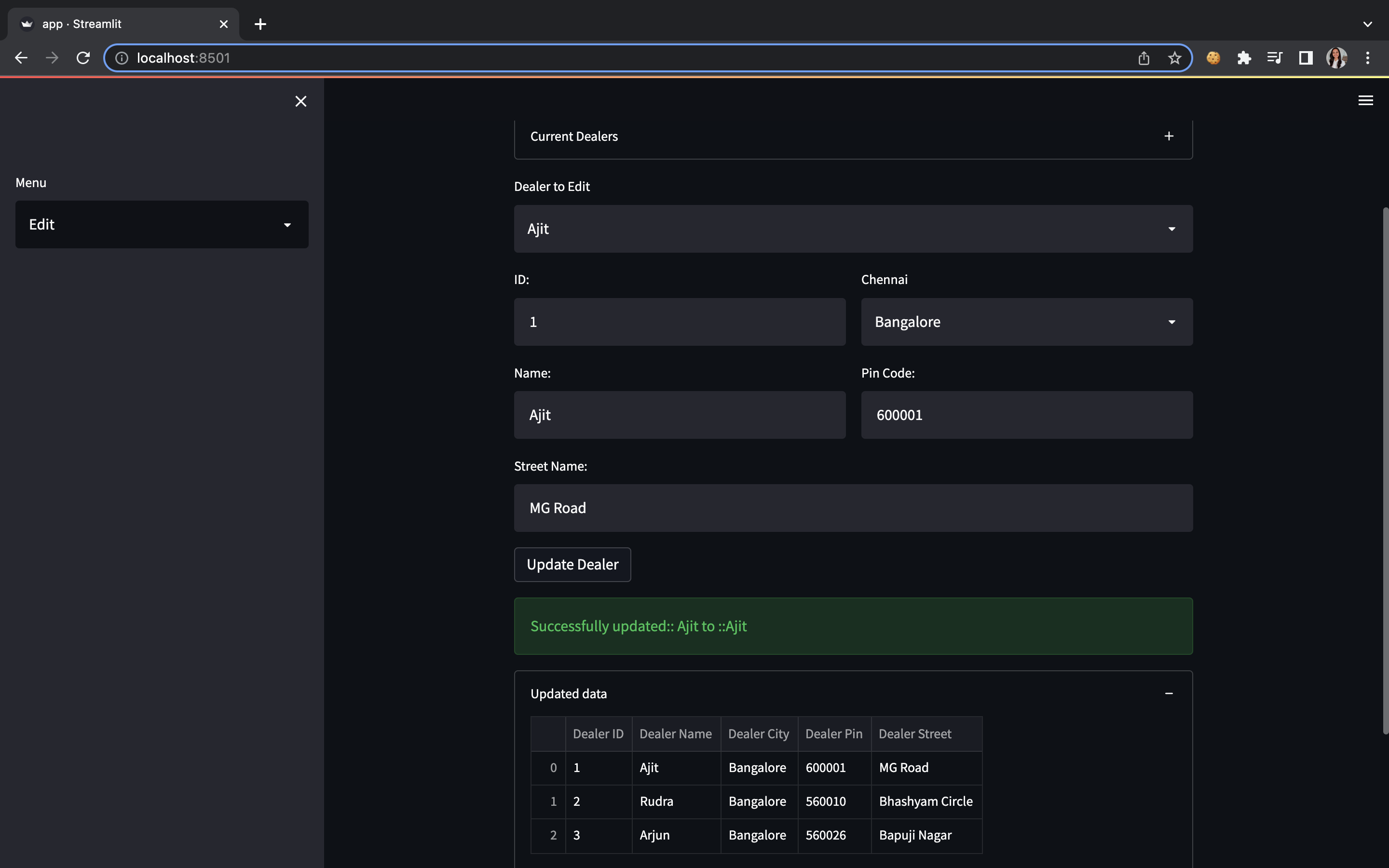
2.



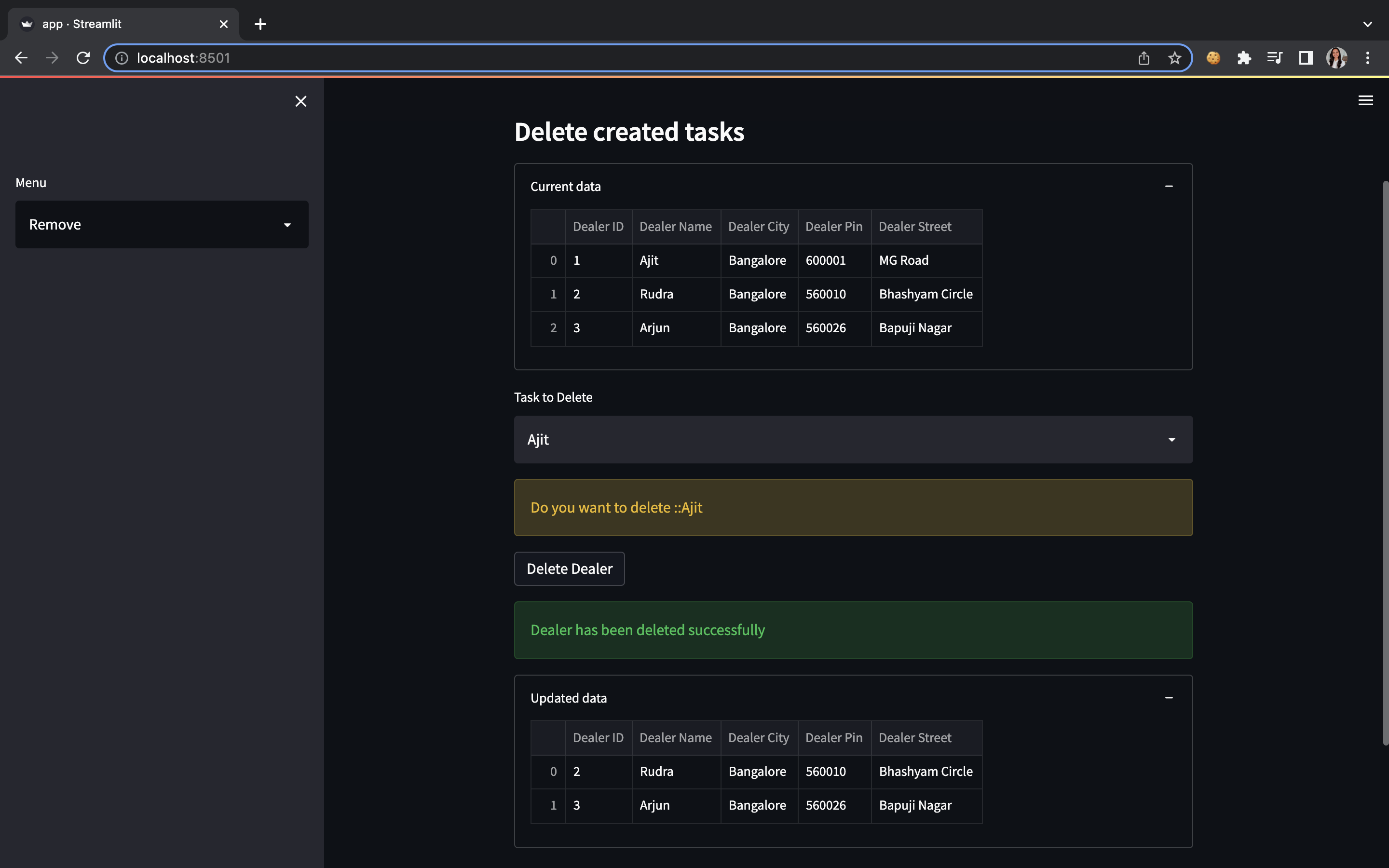
3.



4.



5.



6.