**Design Document**

**Overview**

This document provides a detailed explanation of the code used to filter and display bus routes based on various criteria. The application is built using Python with Streamlit for the user interface and MySQL for data storage and retrieval.

**Code Explanation**

**Import Statements**

import pandas as pd

import streamlit as st

import mysql.connector

from DataClean\_DB\_Insert import create\_connection

* **pandas**: Used for data manipulation and analysis.
* **streamlit**: Used for building the interactive web application.
* **mysql.connector**: Provides MySQL database connectivity.
* **create\_connection**: Custom function from DataClean\_DB\_Insert to establish a database connection.

**Function Definitions**

**get\_state()**

def get\_state():

# Establish a connection to the MySQL database

mydb = create\_connection()

# Create a cursor object

cursor = mydb.cursor()

# Define the query to get distinct states from bus\_routes

query = "SELECT DISTINCT state FROM bus\_routes"

# Execute the query

cursor.execute(query)

# Fetch all the results

states = [row[0] for row in cursor.fetchall()]

# Close the cursor and connection

cursor.close()

mydb.close()

return states

* **Purpose**: Fetches distinct states from the bus\_routes table in the MySQL database.
* **Details**:
  + Establishes a database connection.
  + Executes a SQL query to retrieve distinct states.
  + Returns the list of states.

**get\_route(state=None)**

def get\_route(state=None):

if state is None:

return [] # Return an empty list if no state is provided

# Establish a connection to the MySQL database

mydb = create\_connection()

# Create a cursor object

cursor = mydb.cursor()

# Define the query to get distinct routes from bus\_routes

query = "SELECT DISTINCT route\_name FROM bus\_routes where state = %s"

params = [state]

# Execute the query

cursor.execute(query, tuple(params))

# Fetch all the results

bus\_route = [row[0] for row in cursor.fetchall()]

# Close the cursor and connection

cursor.close()

mydb.close()

return bus\_route

* **Purpose**: Fetches distinct bus routes for a given state from the bus\_routes table.
* **Details**:
  + Checks if a state is provided; if not, returns an empty list.
  + Executes a SQL query to retrieve distinct bus routes for the specified state.
  + Returns the list of bus routes.

**get\_filtered\_data(statename=None, route=None, operator=None, departure\_time=None, bus\_type=None, ratings=None, seats=None, busfare=None)**

def get\_filtered\_data(statename=None, route=None, operator=None, departure\_time=None, bus\_type=None, ratings=None, seats=None, busfare=None):

# Establish a connection to the MySQL database

mydb = create\_connection()

# Create a cursor object

cursor = mydb.cursor()

# Define the base query

query = f"SELECT \* FROM bus\_routes WHERE state = %s"

params = [statename]

# Add filters to the query if they are selected

if operator:

query += " AND operator = %s"

params.append(operator)

if departure\_time:

query += " AND " + departure\_time

if bus\_type:

query += " AND bustype LIKE %s"

params.append(bus\_type)

if ratings:

query += " AND star\_rating " + ratings

if seats:

query += " AND seats\_available " + seats

if route:

query += " AND route\_name = %s"

params.append(route)

if busfare:

query += " AND " + busfare

# Execute the query

cursor.execute(query, tuple(params))

# Fetch all the results

columns = [desc[0] for desc in cursor.description]

results = cursor.fetchall()

# Close the cursor and connection

cursor.close()

mydb.close()

# If no results, return an empty DataFrame

if not results:

return pd.DataFrame(columns=columns)

# Create DataFrame

df = pd.DataFrame(results, columns=columns)

# Check and process 'departing\_time' and 'reaching\_time' if they exist in the DataFrame

for time\_column in ['departing\_time', 'reaching\_time']:

if time\_column in df.columns:

# Extract the time component from the Timedelta and convert it to a string

df[time\_column] = df[time\_column].apply(

lambda x: (pd.to\_timedelta(x).components.hours, pd.to\_timedelta(x).components.minutes))

# Format the time as 'HH:MM:SS'

df[time\_column] = df[time\_column].apply(lambda x: f"{x[0]:02}:{x[1]:02}:00")

return df

* **Purpose**: Retrieves and filters bus route data based on user-selected criteria.
* **Details**:
  + Builds a SQL query dynamically based on the provided filters.
  + Executes the query and fetches the results.
  + Processes the time columns if present.
  + Returns the data as a Pandas DataFrame.

**allfilterfunc()**

def allfilterfunc():

# Mandatory filter starts here

col1, col2 = st.columns(2)

with col1:

state = get\_state()

filter1 = st.selectbox("State Name", options=[""] + state)

with col2:

# Static list for optional filter

optional\_filter = st.selectbox("Bus Operator Pvt/Govt", options=["", "Government", "Private"])

# Step 2: Display additional filters based on initial selection

if filter1:

st.write("Additional Filters")

bus\_route = get\_route(filter1)

col7, col8 = st.columns(2)

with col7:

filter6 = st.selectbox("Bus Route", options=[""] + bus\_route)

with col8:

filter7 = st.selectbox("Bus Fare", options=["", "< 500", "500 - 1000", "> 1000"])

col3, col4 = st.columns(2)

with col3:

filter2 = st.selectbox("Departure Time", options=["", "06:00 - 12:00 Morning", "12:00 - 18:00 Afternoon", "18:00 - 24:00 Evening", "00:00 - 06:00 Night"])

with col4:

filter3 = st.selectbox("Bus Type:", options=["", "Seater", "Sleeper", "AC", "NonAC"])

col5, col6 = st.columns(2)

with col5:

filter4 = st.selectbox("Travellers Ratings", options=["", "4 \* & Above", "3 \* To 4 \*", "Below 3 \*"])

with col6:

filter5 = st.selectbox("Seats Availability", options=["", "Less than 4", "More than 4"])

# Mapping filter3 to corresponding SQL conditions

bus\_fare = None

if filter7 == "< 500":

bus\_fare = "price < 500"

elif filter7 == "500 - 1000":

bus\_fare = "price BETWEEN '500' AND '1000'"

elif filter7 == "> 1000":

bus\_fare = "price > 1000"

# Mapping filter2 to corresponding SQL conditions

DepartureCond = None

if filter2 == "06:00 - 12:00 Morning":

DepartureCond = "TIME(departing\_time) BETWEEN '06:00:00' AND '12:00:00'"

elif filter2 == "12:00 - 18:00 Afternoon":

DepartureCond = "TIME(departing\_time) BETWEEN '12:00:00' AND '18:00:00'"

elif filter2 == "18:00 - 24:00 Evening":

DepartureCond = "TIME(departing\_time) BETWEEN '18:00:00' AND '24:00:00'"

elif filter2 == "00:00 - 06:00 Night":

DepartureCond = "TIME(departing\_time) BETWEEN '00:00:00' AND '06:00:00'"

# Mapping filter3 to corresponding SQL conditions

BusTypeCond = None

if filter3 == "Seater":

BusTypeCond = "%Seater%"

elif filter3 == "Sleeper":

BusTypeCond = "%Sleeper%"

elif filter3 == "AC":

BusTypeCond = "%A/C%"

elif filter3 == "NonAC":

BusTypeCond = "%Non AC%"

# Mapping filter4 to corresponding SQL conditions

ratings\_cond = None

if filter4 == "4 \* & Above":

ratings\_cond = ">= 4"

elif filter4 == "3 \* To 4 \*":

ratings\_cond = "BETWEEN 3 AND 4"

elif filter4 == "Below 3 \*":

ratings\_cond = "< 3"

# Mapping filter5 to corresponding SQL conditions

Seats\_avail = None

if filter5 == "Less than 4":

Seats\_avail = "<= 4"

elif filter5 == "More than 4":

Seats\_avail = "> 4"

route1 = filter6

# Step 4: Display the "Search" button

if st.button("Search"):

st.subheader("Filtered Results")

filtered\_df = get\_filtered\_data(

statename=filter1, # Use the selected state

route=route1,

operator=optional\_filter,

departure\_time=DepartureCond,

bus\_type=BusTypeCond,

ratings=ratings\_cond,

seats=Seats\_avail,

busfare=bus\_fare

)

# Check if the DataFrame is empty and display a message

if filtered\_df.empty:

st.write("No results found for the selected filters.")

else:

# Display the filtered DataFrame

st.dataframe(filtered\_df)

* **Purpose**: Defines the user interface and filters for the Streamlit application.
* **Details**:
  + Displays mandatory filters (state and bus operator).
  + Shows additional filters based on the state selected by the user.
  + Maps selected filter values to corresponding SQL conditions.
  + Displays the filtered results or a message if no results are found.

**Conclusion**

This document provides an in-depth look at the code used for filtering and displaying bus route data. It outlines the purpose and details of each function, explaining how they contribute to the overall functionality of the Streamlit application.