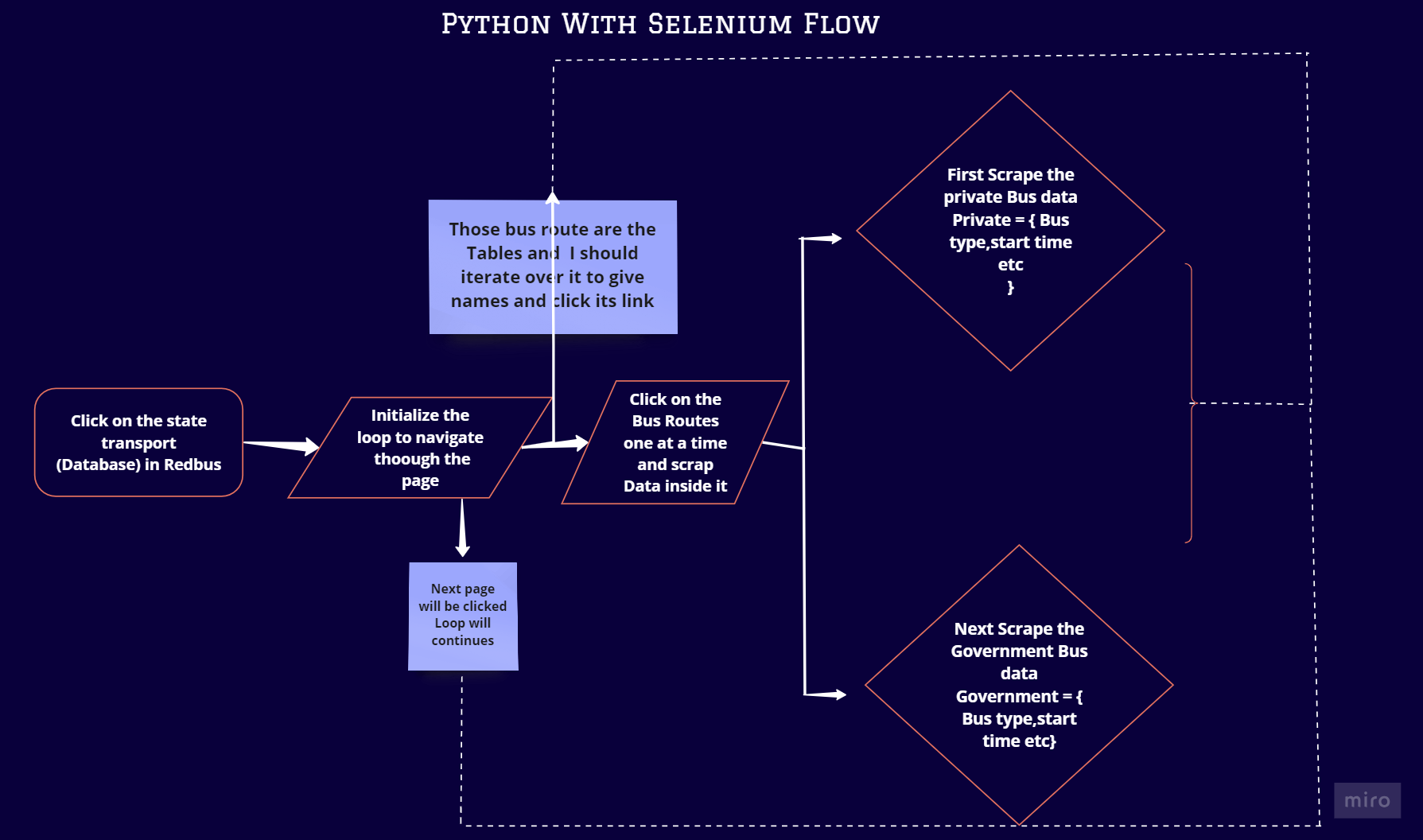
Redbus Project Documentation

Note : Please try to open it in Google docs for better visibility.

**Python code flow for Scraping the data from the Redbus website**



# Python Complete code:

import json

import time

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import (

ElementNotInteractableException,

TimeoutException,

ElementClickInterceptedException,

NoSuchElementException

)

class Redbus:

def \_\_init\_\_(self, Xpath):

self.Xpath = Xpath

self.Bus = {}

# Initialize the Chrome driver

self.driver = webdriver.Chrome()

# Open the Redbus page

self.driver.get('https://www.redbus.in/')

time.sleep(5)

# Scroll horizontally to bring the element into view

target\_element = WebDriverWait(self.driver, 20).until(EC.presence\_of\_element\_located((By.XPATH, self.Xpath)))

self.driver.execute\_script("arguments[0].scrollIntoView({block: 'center', inline: 'center'});", target\_element)

time.sleep(2)

# Click on the state bus link

WebDriverWait(self.driver, 20).until(EC.element\_to\_be\_clickable((By.XPATH, self.Xpath))).click()

time.sleep(10)

Bus\_Route\_link = []

Bus\_Route\_name = []

# Loop to gather bus routes and their links from all pages until no more pages are available

page = 1

while True:

try:

current\_page\_links = [i.get\_attribute('href') for i in self.driver.find\_elements(By.XPATH, "//div[@class='route\_details']//a")]

current\_page\_names = [i.text for i in self.driver.find\_elements(By.XPATH, "//a[@class='route']")]

# Only add unique links and names to the main list

for link, name in zip(current\_page\_links, current\_page\_names):

if link not in Bus\_Route\_link:

Bus\_Route\_link.append(link)

Bus\_Route\_name.append(name)

# Print the bus routes and links to verify they are being collected

print(f"Page {page} - Bus Routes and Links:")

for name, link in zip(Bus\_Route\_name, Bus\_Route\_link):

print(f"Route Name: {name}, Route Link: {link}")

print("\n")

# Try to navigate to the next page

next\_page\_xpath = f"//div[12]/div[{page + 1}]"

if not self.driver.find\_elements(By.XPATH, next\_page\_xpath):

print(f"Page {page + 1} does not exist. Exiting loop.")

break

# Wait for the element to be present and visible

element = WebDriverWait(self.driver, 25).until(EC.visibility\_of\_element\_located((By.XPATH, next\_page\_xpath)))

# Scroll the element into view before clicking

self.driver.execute\_script("arguments[0].scrollIntoView(true);", element)

time.sleep(1)

# Attempt to click the element

element.click()

time.sleep(5)

page += 1

except ElementNotInteractableException as e:

print(f"Error navigating to page {page + 1}: {e}")

break

except ElementClickInterceptedException as e:

print(f"Click intercepted when trying to navigate to page {page + 1}: {e}")

break

except (TimeoutException, NoSuchElementException) as e:

print(f"Error loading page {page + 1}: {e}")

break

def smooth\_scroll():

"""Function to scroll to the bottom of the page to ensure all elements are loaded."""

last\_height = self.driver.execute\_script("return document.body.scrollHeight")

while True:

# Scroll down to the bottom

self.driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

# Wait for new content to load

time.sleep(2)

# Calculate new scroll height and compare with last height

new\_height = self.driver.execute\_script("return document.body.scrollHeight")

if new\_height == last\_height:

break

last\_height = new\_height

# Start scraping bus details from each link

for name, link in zip(Bus\_Route\_name, Bus\_Route\_link):

self.driver.get(link)

time.sleep(10)

self.Bus[name] = {"Private": {}, "Government": {}}

# Wait for elements to load and fetch private bus details

time.sleep(10)

try:

# Smooth scroll before scraping private bus details

smooth\_scroll()

# Scrape private bus details

Bus\_travel\_Name, Bus\_Confort\_Type, Bus\_start\_time, Bus\_end\_time, Total\_travel\_time, Rating, Seat\_availability, Price, Reach\_date = WebDriverWait(self.driver, 25).until(

lambda driver: (

driver.find\_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']"),

driver.find\_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']"),

driver.find\_elements(By.XPATH, "//div[@class='dp-time f-19 d-color f-bold']"),

driver.find\_elements(By.XPATH, "//div[@class='bp-time f-19 d-color disp-Inline']"),

driver.find\_elements(By.XPATH, "//div[@class='dur l-color lh-24']"),

driver.find\_elements(By.XPATH, "//div[@class='rating-sec lh-24']//span"),

driver.find\_elements(By.XPATH, "//div[@class='seat-left m-top-30']"),

driver.find\_elements(By.XPATH, "//div[@class='fare d-block']"),

driver.find\_elements(By.XPATH, "//div[@class='next-day-dp-lbl m-top-16']")

)

)

# Print the bus details to verify they are being scraped

print(f"Scraped data for route: {name}")

print("Bus travel Name:", [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != ''])

print("Bus Comfort Type:", [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != ''])

print("Bus Start Time:", [elem.text.strip() for elem in Bus\_start\_time if elem.text != ''])

print("Bus End Time:", [elem.text.strip() for elem in Bus\_end\_time if elem.text != ''])

print("Total Travel Time:", [elem.text.strip() for elem in Total\_travel\_time if elem.text != ''])

print("Rating:", [float(elem.text.strip()) for elem in Rating if elem.text != ''])

print("Seat Availability:", [int(i.text.split()[0]) for i in Seat\_availability if i.text != ''])

print("Price:", [i.text[3:].strip() for i in Price if i.text[3:] != ''])

print("Reach Date:", [elem.text.strip() for elem in Reach\_date if elem.text != ''])

print("\n")

# Store scraped data for private buses

self.Bus[name]["Private"]["Bus\_Name"] = [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != '']

self.Bus[name]["Private"]["Bus\_Type"] = [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != '']

self.Bus[name]["Private"]["Departing\_Time"] = [elem.text.strip() for elem in Bus\_start\_time if elem.text != '']

self.Bus[name]["Private"]["Reaching\_Time"] = [elem.text.strip() for elem in Bus\_end\_time if elem.text != '']

self.Bus[name]["Private"]["Duration"] = [elem.text.strip() for elem in Total\_travel\_time if elem.text != '']

self.Bus[name]["Private"]["Star\_Rating"] = [float(elem.text.strip()) for elem in Rating if elem.text != '']

self.Bus[name]["Private"]["Seat\_availability"] = [int(i.text.split()[0]) for i in Seat\_availability if i.text != '']

self.Bus[name]["Private"]["Price"] = [i.text[3:].strip() for i in Price if i.text[3:] != '']

self.Bus[name]["Private"]["Reach\_date"] = [elem.text.strip() for elem in Reach\_date if elem.text != '']

# Click to switch to government buses

try:

button = WebDriverWait(self.driver, 15).until(EC.element\_to\_be\_clickable((By.XPATH, "//div[@class='button']")))

# Scroll the button into view

self.driver.execute\_script("arguments[0].scrollIntoView(true);", button)

time.sleep(1)

self.driver.execute\_script("arguments[0].click();", button)

except ElementClickInterceptedException as e:

print(f"Error clicking the button to switch to government buses: {e}")

# Smooth scroll before scraping government bus details

smooth\_scroll()

# Scrape government bus details

Bus\_travel\_Name, Bus\_Confort\_Type, Bus\_start\_time, Bus\_end\_time, Total\_travel\_time, Rating, Seat\_availability, Price, Reach\_date = WebDriverWait(self.driver, 25).until(

lambda driver: (

driver.find\_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']"),

driver.find\_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']"),

driver.find\_elements(By.XPATH, "//div[@class='dp-time f-19 d-color f-bold']"),

driver.find\_elements(By.XPATH, "//div[@class='bp-time f-19 d-color disp-Inline']"),

driver.find\_elements(By.XPATH, "//div[@class='dur l-color lh-24']"),

driver.find\_elements(By.XPATH, "//div[@class='rating-sec lh-24']//span"),

driver.find\_elements(By.XPATH, "//div[@class='seat-left m-top-30']"),

driver.find\_elements(By.XPATH, "//div[@class='fare d-block']"),

driver.find\_elements(By.XPATH, "//div[@class='next-day-dp-lbl m-top-16']")

)

)

# Print the bus details to verify they are being scraped

print(f"Scraped data for government buses on route: {name}")

print("Bus travel Name:", [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != ''])

print("Bus Comfort Type:", [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != ''])

print("Bus Start Time:", [elem.text.strip() for elem in Bus\_start\_time if elem.text != ''])

print("Bus End Time:", [elem.text.strip() for elem in Bus\_end\_time if elem.text != ''])

print("Total Travel Time:", [elem.text.strip() for elem in Total\_travel\_time if elem.text != ''])

print("Rating:", [float(elem.text.strip()) for elem in Rating if elem.text != ''])

print("Seat Availability:", [int(i.text.split()[0]) for i in Seat\_availability if i.text != ''])

print("Price:", [i.text[3:].strip() for i in Price if i.text[3:] != ''])

print("Reach Date:", [elem.text.strip() for elem in Reach\_date if elem.text != ''])

print("\n")

# Store scraped data for government buses

self.Bus[name]["Government"]["Bus\_Name"] = [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != '']

self.Bus[name]["Government"]["Bus\_Type"] = [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != '']

self.Bus[name]["Government"]["Departing\_Time"] = [elem.text.strip() for elem in Bus\_start\_time if elem.text != '']

self.Bus[name]["Government"]["Reaching\_Time"] = [elem.text.strip() for elem in Bus\_end\_time if elem.text != '']

self.Bus[name]["Government"]["Duration"] = [elem.text.strip() for elem in Total\_travel\_time if elem.text != '']

self.Bus[name]["Government"]["Star\_Rating"] = [float(elem.text.strip()) for elem in Rating if elem.text != '']

self.Bus[name]["Government"]["Seat\_availability"] = [int(i.text.split()[0]) for i in Seat\_availability if i.text != '']

self.Bus[name]["Government"]["Price"] = [i.text[3:].strip() for i in Price if i.text[3:] != '']

self.Bus[name]["Government"]["Reach\_date"] = [elem.text.strip() for elem in Reach\_date if elem.text != '']

except TimeoutException as e:

print(f"Error fetching bus details for {name}: {e}")

# Close the WebDriver

self.driver.quit()

def get\_bus\_data(self):

"""Return the scraped bus data."""

return self.Bus

# Creating an object of Redbus

Buses = Redbus("//div[@class='rtcNameMain']/div[@class='rtcName' and text()='WBSTC']")

Value = Buses.get\_bus\_data()

## 

## **Import Section:**

Imported various modules based on the requirement and time module is for the implementing the sleep in between the page or sessions navigations and also added a exceptions handling for catching the various error and run without any errors

import json

import time

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from selenium.common.exceptions import (

ElementNotInteractableException,

TimeoutException,

ElementClickInterceptedException,

NoSuchElementException

## **Defining the class and getting into the redbus website:**

Creating the class for the code reusability and defining it with the name redbus it will take only one argument which is the path to clicking the right state transport page in red bus website and self.bus will store the final scraped Bus data it will first get into the <https://www.redbus.in> page and scroll till the text visible it will click the respective page that I have given as a argument

class Redbus:

def \_\_init\_\_(self, Xpath):

self.Xpath = Xpath

self.Bus = {} # Storing the Bus dictionary as an instance variable

# Initialize the Chrome driver

self.driver = webdriver.Chrome()

# Open the Redbus page

self.driver.get('https://www.redbus.in/')

time.sleep(5)

# Scroll horizontally to bring the element into view

target\_element = WebDriverWait(self.driver, 20).until(EC.presence\_of\_element\_located((By.XPATH, self.Xpath)))

self.driver.execute\_script("arguments[0].scrollIntoView({block: 'center', inline: 'center'});", target\_element)

time.sleep(2)

# Click on the state bus link

WebDriverWait(self.driver, 20).until(EC.element\_to\_be\_clickable((By.XPATH, self.Xpath))).click()

time.sleep(10)

## 

## **Creating the Scroll function and storing the Bus route name and Bus route link for accessing it:**

This Bus\_Route\_link and Bus\_Route\_name will store the first page name and link and i have included a Smooth\_scroll function because without this it only take first 5 to 6 data only so I include this after this the page is scrolled smoothly and i can able to scrape all the data I want

def smooth\_scroll():

"""Function to scroll to the bottom of the page to ensure all elements are loaded."""

last\_height = self.driver.execute\_script("return document.body.scrollHeight")

while True:

# Scroll down to the bottom

self.driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);")

# Wait for new content to load

time.sleep(2)

# Calculate new scroll height and compare with last height

new\_height = self.driver.execute\_script("return document.body.scrollHeight")

if new\_height == last\_height:

break

last\_height = new\_height

## 

## **Page Navigation loop and to scrape the Bus details:**

Bus\_Route\_link = []

Bus\_Route\_name = []

# Loop to gather bus routes and their links from all pages until no more pages are available

page = 1

while True:

try:

current\_page\_links = [i.get\_attribute('href') for i in self.driver.find\_elements(By.XPATH, "//div[@class='route\_details']//a")]

current\_page\_names = [i.text for i in self.driver.find\_elements(By.XPATH, "//a[@class='route']")]

# Only add unique links and names to the main list

for link, name in zip(current\_page\_links, current\_page\_names):

if link not in Bus\_Route\_link:

Bus\_Route\_link.append(link)

Bus\_Route\_name.append(name)

# Print the bus routes and links to verify they are being collected

print(f"Page {page} - Bus Routes and Links:")

for name, link in zip(Bus\_Route\_name, Bus\_Route\_link):

print(f"Route Name: {name}, Route Link: {link}")

print("\n")

# Try to navigate to the next page

next\_page\_xpath = f"//div[12]/div[{page + 1}]"

if not self.driver.find\_elements(By.XPATH, next\_page\_xpath):

print(f"Page {page + 1} does not exist. Exiting loop.")

break

# Wait for the element to be present and visible

element = WebDriverWait(self.driver, 25).until(EC.visibility\_of\_element\_located((By.XPATH, next\_page\_xpath)))

# Scroll the element into view before clicking

self.driver.execute\_script("arguments[0].scrollIntoView(true);", element)

time.sleep(1)

# Attempt to click the element

element.click()

time.sleep(5)

page += 1

except ElementNotInteractableException as e:

print(f"Error navigating to page {page + 1}: {e}")

break

except ElementClickInterceptedException as e:

print(f"Click intercepted when trying to navigate to page {page + 1}: {e}")

break

except (TimeoutException, NoSuchElementException) as e:

print(f"Error loading page {page + 1}: {e}")

break

In this it will go to the first page and scrape all the link and the Bus route name and it will append it into the empty list and it click the second page and it will scrape all the link and name similarly for all the page after scraped all the link and name it will start scraping the data.

In try block the process will repeat and scroll to view the page number if the page is there means it will click if it was not there means it will break the loop and quit.

In exception block it has ElementNotInteractableException,ElementClickInterceptedException TimeoutException and NoSuchElementException so the program runs flawlessly it will have the error with stopping the program

## Private Bus data scraping

In this block the Bus\_Route\_name and Bus\_Route\_link is zip together to access it simultaneously in the dictionary is Bus name is a key in side there are two nested key Private and Government inside that there are nested key have various bus details that is scraped from the redbus website the private bus details is scraped using this code and the sleep time is also optimized so it will start scraping after page loaded and lambda function is used to point out or map the path to the variable and store the scraped data in that variable by that specific path while before storing it inside a Private dictionary it was extracted properly using list comprehension as per the needs so the Private bus detail is scraped successfully

for name, link in zip(Bus\_Route\_name, Bus\_Route\_link):

self.driver.get(link)

time.sleep(10)

self.Bus[name] = {"Private": {}, "Government": {}}

# Wait for elements to load and fetch private bus details

time.sleep(10) # Adjust as needed

try:

# Smooth scroll before scraping private bus details

smooth\_scroll()

# Scrape private bus details

Bus\_travel\_Name, Bus\_Confort\_Type, Bus\_start\_time, Bus\_end\_time, Total\_travel\_time, Rating, Seat\_availability, Price, Reach\_date = WebDriverWait(self.driver, 25).until(

lambda driver: (

driver.find\_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']"),

driver.find\_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']"),

driver.find\_elements(By.XPATH, "//div[@class='dp-time f-19 d-color f-bold']"),

driver.find\_elements(By.XPATH, "//div[@class='bp-time f-19 d-color disp-Inline']"),

driver.find\_elements(By.XPATH, "//div[@class='dur l-color lh-24']"),

driver.find\_elements(By.XPATH, "//div[@class='rating-sec lh-24']//span"),

driver.find\_elements(By.XPATH, "//div[@class='seat-left m-top-30']"),

driver.find\_elements(By.XPATH, "//div[@class='fare d-block']"),

driver.find\_elements(By.XPATH, "//div[@class='next-day-dp-lbl m-top-16']")

)

)

# Store scraped data for private buses

self.Bus[name]["Private"]["Bus\_Name"] = [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != '']

self.Bus[name]["Private"]["Bus\_Type"] = [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != '']

self.Bus[name]["Private"]["Departing\_Time"] = [elem.text.strip() for elem in Bus\_start\_time if elem.text != '']

self.Bus[name]["Private"]["Reaching\_Time"] = [elem.text.strip() for elem in Bus\_end\_time if elem.text != '']

self.Bus[name]["Private"]["Duration"] = [elem.text.strip() for elem in Total\_travel\_time if elem.text != '']

self.Bus[name]["Private"]["Star\_Rating"] = [float(elem.text.strip()) for elem in Rating if elem.text != '']

self.Bus[name]["Private"]["Seat\_availability"] = [int(i.text.split()[0]) for i in Seat\_availability if i.text != '']

self.Bus[name]["Private"]["Price"] = [i.text[3:].strip() for i in Price if i.text[3:] != '']

self.Bus[name]["Private"]["Reach\_date"] = [elem.text.strip() for elem in Reach\_date if elem.text != '']

## Government Bus data scraping:

After completion and storing the private bus data it will scrape the government bus data first it will click if there is any error means it will catch and try again

Bus name is a key in side there are two nested key Private and Government inside that there are nested key have various bus details that is scraped from the redbus website the Government bus details is scraped using this code and the sleep time is also optimized so it will start scraping after page loaded and lambda function is used to point out or map the path to the variable and store the scraped data in that variable by that specific path while before storing it inside a Government dictionary it was extracted properly using list comprehension as per the needs so the Government bus detail is scraped successfully

It will also handel the page not found error so that even if there is no bus available at the time of scraping also it will scrape without any error it will just skips that particular bus details

try:

button = WebDriverWait(self.driver, 15).until(EC.element\_to\_be\_clickable((By.XPATH, "//div[@class='button']")))

# Scroll the button into view

self.driver.execute\_script("arguments[0].scrollIntoView(true);", button)

time.sleep(1) # Optional: Wait a moment for any animations or loading to settle

# Click the button using JavaScript as a fallback

self.driver.execute\_script("arguments[0].click();", button)

except ElementClickInterceptedException as e:

print(f"Error clicking the button to switch to government buses: {e}")

# Smooth scroll before scraping government bus details

smooth\_scroll()

# Scrape government bus details

Bus\_travel\_Name, Bus\_Confort\_Type, Bus\_start\_time, Bus\_end\_time, Total\_travel\_time, Rating, Seat\_availability, Price, Reach\_date = WebDriverWait(self.driver, 25).until(

lambda driver: (

driver.find\_elements(By.XPATH, "//div[@class='travels lh-24 f-bold d-color']"),

driver.find\_elements(By.XPATH, "//div[@class='bus-type f-12 m-top-16 l-color evBus']"),

driver.find\_elements(By.XPATH, "//div[@class='dp-time f-19 d-color f-bold']"),

driver.find\_elements(By.XPATH, "//div[@class='bp-time f-19 d-color disp-Inline']"),

driver.find\_elements(By.XPATH, "//div[@class='dur l-color lh-24']"),

driver.find\_elements(By.XPATH, "//div[@class='rating-sec lh-24']//span"),

driver.find\_elements(By.XPATH, "//div[@class='seat-left m-top-30']"),

driver.find\_elements(By.XPATH, "//div[@class='fare d-block']"),

driver.find\_elements(By.XPATH, "//div[@class='next-day-dp-lbl m-top-16']")

)

)

# Store scraped data for government buses

self.Bus[name]["Government"]["Bus\_Name"] = [elem.text.strip() for elem in Bus\_travel\_Name if elem.text != '']

self.Bus[name]["Government"]["Bus\_Type"] = [elem.text.strip() for elem in Bus\_Confort\_Type if elem.text != '']

self.Bus[name]["Government"]["Departing\_Time"] = [elem.text.strip() for elem in Bus\_start\_time if elem.text != '']

self.Bus[name]["Government"]["Reaching\_Time"] = [elem.text.strip() for elem in Bus\_end\_time if elem.text != '']

self.Bus[name]["Government"]["Duration"] = [elem.text.strip() for elem in Total\_travel\_time if elem.text != '']

self.Bus[name]["Government"]["Star\_Rating"] = [float(elem.text.strip()) for elem in Rating if elem.text != '']

self.Bus[name]["Government"]["Seat\_availability"] = [int(i.text.split()[0]) for i in Seat\_availability if i.text != '']

self.Bus[name]["Government"]["Price"] = [i.text[3:].strip() for i in Price if i.text[3:] != '']

self.Bus[name]["Government"]["Reach\_date"] = [elem.text.strip() for elem in Reach\_date if elem.text != '']

except TimeoutException as e:

print(f"Error fetching bus details for {name}: {e}")

## 

## 

## Passing the argument inside the class and getting the data:

Finally I can pass the text of the bus that I need to scrape and the program will scrape the data and I can access the data by the method get\_bus\_data with an object Buses

def get\_bus\_data(self):

"""Return the scraped bus data."""

return self.Bus

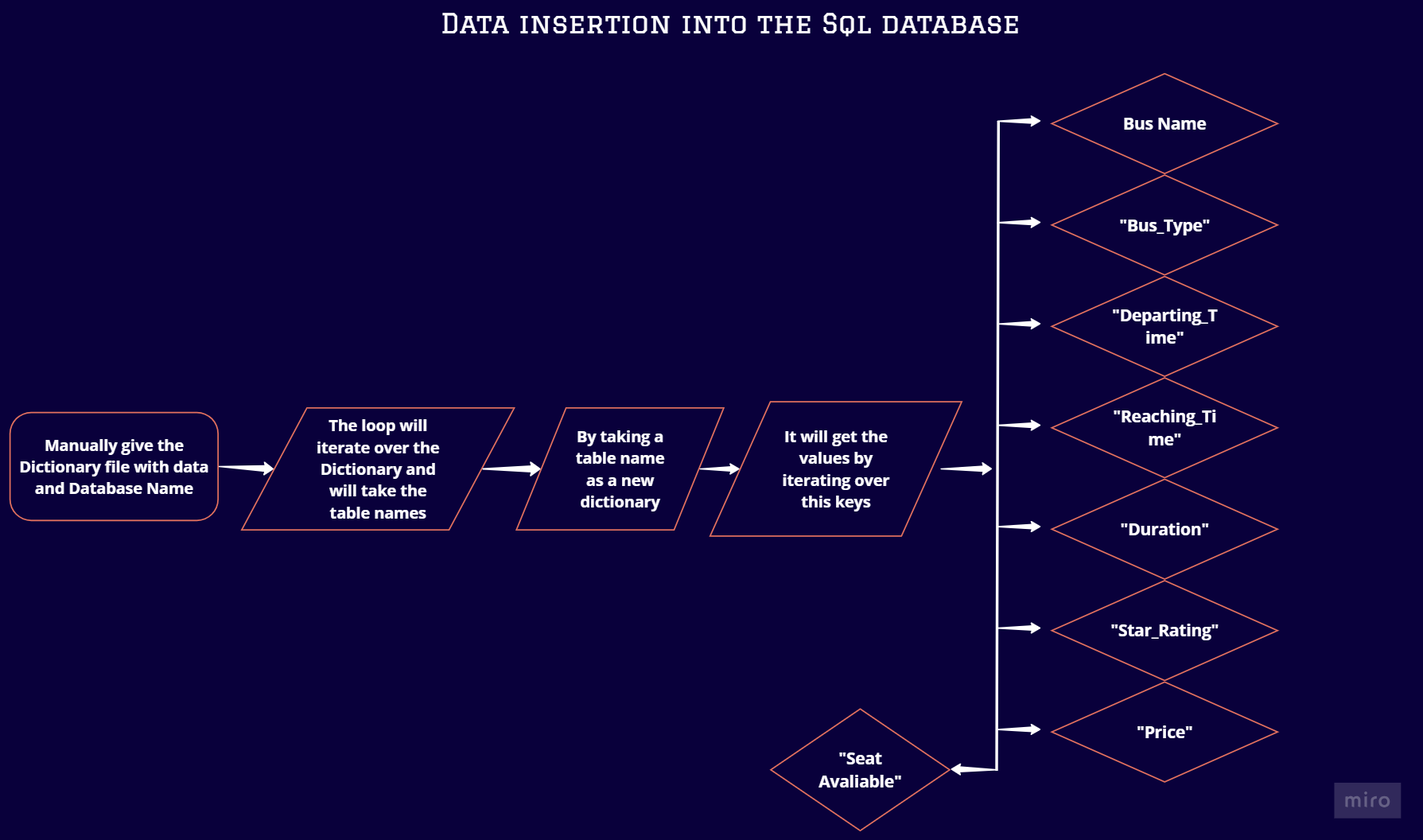
# Creating an instance of Redbus

Buses = Redbus("//div[@class='rtcNameMain']/div[@class='rtcName' and text()='JKSRTC']")

Value = Buses.get\_bus\_data()

This is how I scraped the data from the redbus website. Let's go to the importing part of the data I have collected and import it into the Sql database effectively.

SQL data collection flow:



# 

# SQL Importing Complete Code:

import mysql.connector

class Importing:

def \_\_init\_\_(self, Bus\_state\_name, Table\_insert, table\_name, database\_name):

self.Bus\_state\_name = Bus\_state\_name

self.Table\_insert = Table\_insert

self.table\_name = table\_name

self.database\_name = database\_name

# Connect to MySQL

self.mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

autocommit=True # Enable autocommit

)

print(self.mydb)

self.mycursor = self.mydb.cursor(buffered=True)

# Create the database

self.mycursor.execute(f"CREATE DATABASE IF NOT EXISTS {self.database\_name}")

self.mycursor.execute(f"USE {self.database\_name}") # Switch to the new database

# Escape table names with backticks to handle spaces and reserved keywords

table\_name\_1 = f"`{self.table\_name}`"

# Create the table with the specified schema, including Bus\_route\_name

self.mycursor.execute(f"""

CREATE TABLE IF NOT EXISTS {table\_name\_1} (

id INT NOT NULL AUTO\_INCREMENT,

Bus\_state\_name VARCHAR(100),

Bus\_route\_name VARCHAR(100),

Bus\_Operator\_type VARCHAR(15),

BusName VARCHAR(50),

BusType VARCHAR(50),

Departing\_Time TIME,

Duration VARCHAR(30),

Reaching\_Time TIME,

Star\_rating FLOAT(5),

Price FLOAT(10),

Seats\_available INT(5),

PRIMARY KEY (id)

)

""")

# Insert data into the table

for bus\_route\_name, operator\_data in self.Table\_insert.items():

for operator\_type in ['Private', 'Government']:

print(f"Inserting data for operator type: {operator\_type} on route: {bus\_route\_name}")

try:

bus\_name = operator\_data[operator\_type]['Bus\_Name']

bus\_type = operator\_data[operator\_type]['Bus\_Type']

departing\_time = operator\_data[operator\_type]['Departing\_Time']

duration = operator\_data[operator\_type]['Duration']

reaching\_time = operator\_data[operator\_type]['Reaching\_Time']

star\_rating = operator\_data[operator\_type]['Star\_Rating']

price = operator\_data[operator\_type]['Price']

seats\_available = operator\_data[operator\_type]['Seat\_availability']

except KeyError as e:

print(f"KeyError: {e} in route {bus\_route\_name} for operator type {operator\_type}. Skipping...")

continue

insert\_query = f"INSERT INTO {table\_name\_1} (Bus\_state\_name, Bus\_route\_name, Bus\_Operator\_type, BusName, BusType, Departing\_Time, Duration, Reaching\_Time, Star\_rating, Price, Seats\_available) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,%s)"

# Get the minimum length among the lists to avoid index errors

value\_lengths = [len(i) for i in [bus\_name, bus\_type, departing\_time, duration, reaching\_time, star\_rating, price, seats\_available]]

min\_length = min(value\_lengths)

print(f"Minimum length of lists: {min\_length}")

# Insert each value in the list as a separate row

for i in range(min\_length):

print(f"Inserting row {i+1} for {bus\_route\_name} - {operator\_type}")

print((

self.Bus\_state\_name,

bus\_route\_name,

operator\_type,

bus\_name[i],

bus\_type[i],

departing\_time[i],

duration[i],

reaching\_time[i],

star\_rating[i],

price[i],

seats\_available[i]

))

self.mycursor.execute(insert\_query, (

self.Bus\_state\_name,

bus\_route\_name,

operator\_type,

bus\_name[i],

bus\_type[i],

departing\_time[i],

duration[i],

reaching\_time[i],

star\_rating[i],

price[i],

seats\_available[i]

))

self.mydb.commit() # Commit after each insertion

#Inputing the arguments State bus name,Variable containing scraped data,Table name,Database name

importer = Importing('WBSTC-West Bengal State Transport',WBSTC, "Bus\_Data", "RedBus")

## Import section and Defining the class:

Here I was importing the mysql connection for connecting the my sql server and i have taking four thing as a Input the arguments State bus name,Variable containing scraped data,Table name,Database name which will insert the data after creating the database and the table and state bus name we need to give it as a input for accessing it furture

import mysql.connector

class Importing:

def \_\_init\_\_(self, Bus\_state\_name, Table\_insert, table\_name, database\_name):

self.Bus\_state\_name = Bus\_state\_name

self.Table\_insert = Table\_insert

self.table\_name = table\_name

self.database\_name = database\_name

# Connect to MySQL

self.mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

autocommit=True # Enable autocommit

)

print(self.mydb)

self.mycursor = self.mydb.cursor(buffered=True)

# Create the database

self.mycursor.execute(f"CREATE DATABASE IF NOT EXISTS {self.database\_name}")

self.mycursor.execute(f"USE {self.database\_name}") # Switch to the new database

# Escape table names with backticks to handle spaces and reserved keywords

table\_name\_1 = f"`{self.table\_name}`"

## **Creating a table and defining the data types:**

Since the route contains keywords such as to I have used a backticks to avoid that and given the proper data types as they mentioned.

# Create the table with the specified schema, including Bus\_route\_name

self.mycursor.execute(f"""

CREATE TABLE IF NOT EXISTS {table\_name\_1} (

id INT NOT NULL AUTO\_INCREMENT,

Bus\_state\_name VARCHAR(100),

Bus\_route\_name VARCHAR(100),

Bus\_Operator\_type VARCHAR(15),

BusName VARCHAR(50),

BusType VARCHAR(50),

Departing\_Time TIME,

Duration VARCHAR(30),

Reaching\_Time TIME,

Star\_rating FLOAT(5),

Price FLOAT(10),

Seats\_available INT(5),

PRIMARY KEY (id)

)

""")

**Inserting the data in to the table:**

Now using the loop and the accessing the key inside the dictionary i have inserted a values in a rows and i have also implemented a try-except statement so if there is any key error is found means it won’t shows any error it just skips that element and continue to the next one

# Insert data into the table

for bus\_route\_name, operator\_data in self.Table\_insert.items():

for operator\_type in ['Private', 'Government']:

print(f"Inserting data for operator type: {operator\_type} on route: {bus\_route\_name}")

try:

bus\_name = operator\_data[operator\_type]['Bus\_Name']

bus\_type = operator\_data[operator\_type]['Bus\_Type']

departing\_time = operator\_data[operator\_type]['Departing\_Time']

duration = operator\_data[operator\_type]['Duration']

reaching\_time = operator\_data[operator\_type]['Reaching\_Time']

star\_rating = operator\_data[operator\_type]['Star\_Rating']

price = operator\_data[operator\_type]['Price']

seats\_available = operator\_data[operator\_type]['Seat\_availability']

except KeyError as e:

print(f"KeyError: {e} in route {bus\_route\_name} for operator type {operator\_type}. Skipping...")

continue

insert\_query = f"INSERT INTO {table\_name\_1} (Bus\_state\_name, Bus\_route\_name, Bus\_Operator\_type, BusName, BusType, Departing\_Time, Duration, Reaching\_Time, Star\_rating, Price, Seats\_available) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s,%s)"

## **Inserting the values using the Keys:**

In this the values inside the each keys are regular for some reason so I have decided to go with min value to avoid the index error and for some routes the government buses are empty or not available for that I have used try-except to handle that issue and finally I have given commit to save the process in database and If I given the arguments means the process will starts and complete the insertion process

value\_lengths = [len(i) for i in [bus\_name, bus\_type, departing\_time, duration, reaching\_time, star\_rating, price, seats\_available]]

min\_length = min(value\_lengths)

print(f"Minimum length of lists: {min\_length}")

# Insert each value in the list as a separate row

for i in range(min\_length):

print(f"Inserting row {i+1} for {bus\_route\_name} - {operator\_type}")

print((

self.Bus\_state\_name,

bus\_route\_name,

operator\_type,

bus\_name[i],

bus\_type[i],

departing\_time[i],

duration[i],

reaching\_time[i],

star\_rating[i],

price[i],

seats\_available[i]

))

self.mycursor.execute(insert\_query, (

self.Bus\_state\_name,

bus\_route\_name,

operator\_type,

bus\_name[i],

bus\_type[i],

departing\_time[i],

duration[i],

reaching\_time[i],

star\_rating[i],

price[i],

seats\_available[i]

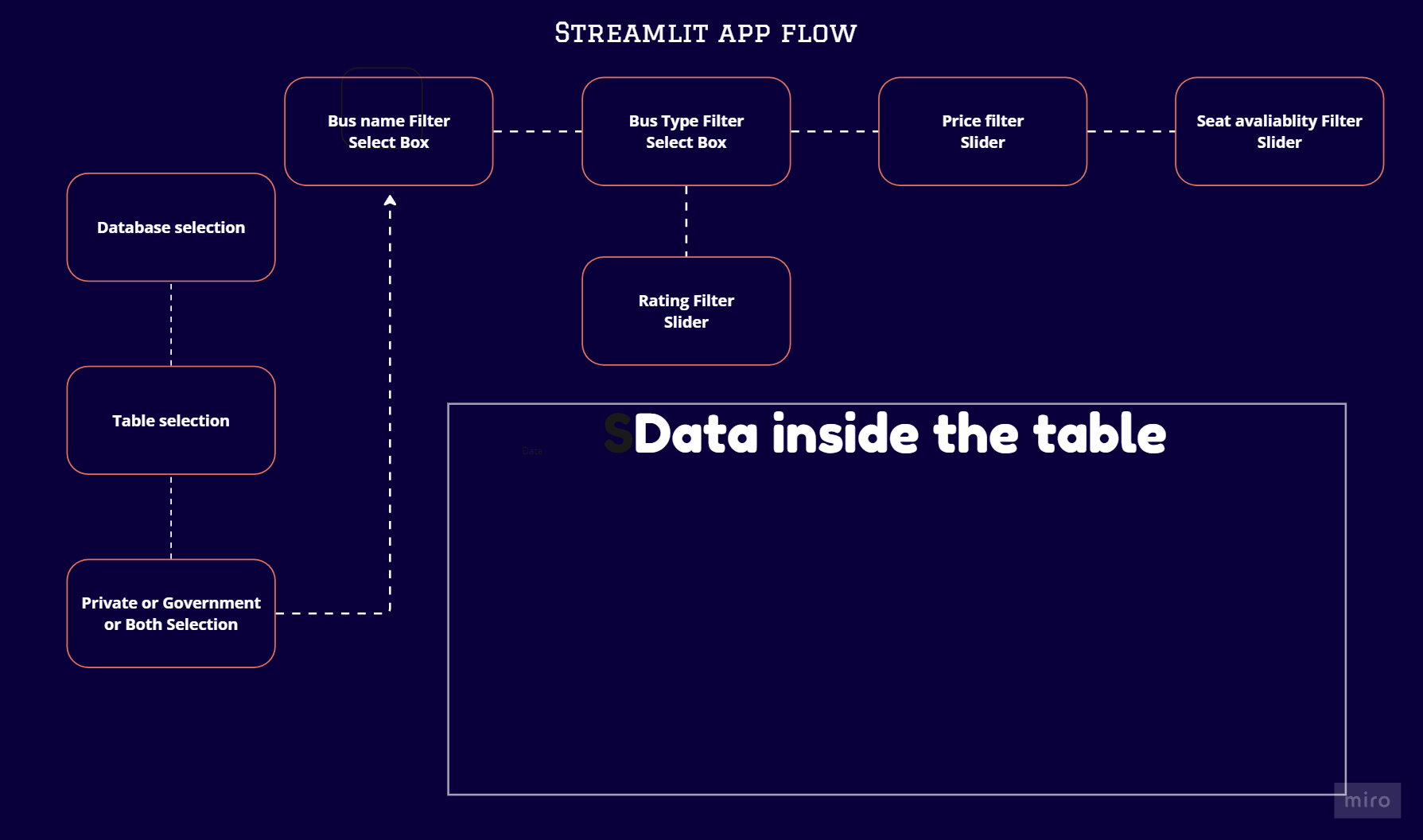
))

self.mydb.commit() # Commit after each insertion

#Inputing the arguments State bus name,Variable containing scraped data,Table name,Database name

importer = Importing('WBSTC-West Bengal State Transport',WBSTC, "Bus\_Data", "RedBus")

**Streamlit view flow and design:**

****

# **Streamlit complete code:**

import streamlit as st

import mysql.connector

import pandas as pd

# MySQL connection setup

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

autocommit=True

)

mycursor = mydb.cursor(buffered=True)

# Fetch list of databases

mycursor.execute("USE Redbus")

# Streamlit UI

st.set\_page\_config(layout='centered')

st.title("Welcome to Bus Selector 🚎🚎")

# Custom CSS for styling and hiding slider labels

st.markdown(

"""

<style>

.sidebar .sidebar-content {

background-color: #004E64;

color: #FFFFFF;

border-right: 2px solid #25A18E;

}

.stSelectbox {

background-color: Orange;

color: #fb6107;

border-radius: 5px;

margin-bottom: 10px;

}

.stSelectbox label {

color: #1f2421;

font-weight: bold;

}

.stButton button:hover {

background-color: #25A18E;

transform: scale(1.05);

}

.stDataFrame, .stTable {

background-color: #E5FFF6;

color: #004E64;

border: 3px solid #004E64;

border-radius:4px;

margin-top: 20px;

}

.stSlider .stSliderLabel {

display: none !important;

}

</style>

""",

unsafe\_allow\_html=True

)

# Sidebar for database and table selection

with st.sidebar:

mycursor.execute(f"SELECT DISTINCT Bus\_state\_name FROM bus\_data")

State\_transport = [state[0] for state in mycursor.fetchall()]

State\_transport = st.selectbox("Select State Transport", State\_transport)

if State\_transport:

mycursor.execute(f"SELECT DISTINCT Bus\_route\_name FROM bus\_data WHERE Bus\_state\_name = '{State\_transport}'")

Bus\_route = [route[0] for route in mycursor.fetchall()]

Bus\_route = st.selectbox("Select Bus route", Bus\_route)

if Bus\_route:

mycursor.execute(f"SELECT DISTINCT Bus\_Operator\_type FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}'")

Select\_operator\_Type = [operator\_type[0] for operator\_type in mycursor.fetchall()]

Select\_operator\_Type.insert(0, "Both")

Select\_operator\_Type = st.selectbox("Bus Operator Types", Select\_operator\_Type, index=0)

# Main content

if State\_transport and Bus\_route and Select\_operator\_Type:

mycursor.execute(f"SELECT DISTINCT BusName FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Name = [column[0] for column in mycursor.fetchall()]

Name.insert(0, "Default") # Add "Default" option to select all

Select\_operator\_Name = st.selectbox("Bus Name", Name, key="bus\_name", index=0)

# Fetch bus types based on selected operator type

mycursor.execute(f"SELECT DISTINCT BusType FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Type = [column[0] for column in mycursor.fetchall()]

Type.insert(0, "Default") # Add "Default" option to select all

Select\_Bus\_Type = st.selectbox("Bus Comfort Type", Type, key="bus\_type", index=0)

# Fetch star ratings

mycursor.execute(f"SELECT Star\_rating FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Star\_Rating = st.slider("Ratings", min\_value=0.0, max\_value=5.0, step=0.1, value=(0.0, 5.0), key="star\_rating")

# Fetch price range

mycursor.execute(f"SELECT Price FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Price = st.slider("Price Range", min\_value=0, max\_value=5000, step=50, value=(0, 5000), key="price")

# Fetch seats available

mycursor.execute(f"SELECT Seats\_available FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Seats\_available = st.slider("Seats Available", min\_value=0, max\_value=60, step=3, value=(0, 60), key="seats\_available")

# Fetch data based on selected filters

with st.spinner('Fetching data...'):

query\_conditions = []

if Select\_operator\_Name != "Default":

query\_conditions.append(f"BusName = '{Select\_operator\_Name}'")

if Select\_Bus\_Type != "Default":

query\_conditions.append(f"BusType = '{Select\_Bus\_Type}'")

query\_conditions.append(f"Star\_rating BETWEEN {Star\_Rating[0]} AND {Star\_Rating[1]}")

query\_conditions.append(f"Price BETWEEN {Price[0]} AND {Price[1]}")

query\_conditions.append(f"Seats\_available BETWEEN {Seats\_available[0]} AND {Seats\_available[1]}")

if Select\_operator\_Type != "Both":

query\_conditions.append(f"Bus\_Operator\_type = '{Select\_operator\_Type}'")

query = f"""

SELECT \*, DATE\_FORMAT(Departing\_Time, '%H:%i:%s') AS Departing\_Time\_, DATE\_FORMAT(Reaching\_Time, '%H:%i:%s') AS Reaching\_Time\_

FROM bus\_data

WHERE Bus\_route\_name = '{Bus\_route}' AND {' AND '.join(query\_conditions)}

"""

mycursor.execute(query)

data = mycursor.fetchall()

columns = mycursor.column\_names

# Convert data to DataFrame without index

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

# Drop old columns

df.drop(['Departing\_Time', 'Reaching\_Time'], axis=1, inplace=True)

# Set 'id' column as the index

df.set\_index('id', inplace=True)

# Display data without index

st.write(f"Buses from {Bus\_route} are")

st.dataframe(df)

## **Import and Title of the app:**

For using the streamlite I have imported it and I have given a sql connection setup for and given a use database name for using the database name ie Redbus in future wherever it needed and finally I have given a page layout as a center and the title.

import streamlit as st

import mysql.connector

import pandas as pd

# MySQL connection setup

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

autocommit=True

)

mycursor = mydb.cursor(buffered=True)

# Fetch list of databases

mycursor.execute("USE Redbus")

# Streamlit UI

st.set\_page\_config(layout='centered')

st.title("Welcome to Bus Selector 🚎🚎")

## **Page look and its code :**

For enhancing the look of the page I have used this code which will make a good first impression.

# Custom CSS for styling

st.markdown(

"""

<style>

.sidebar .sidebar-content {

background-color: #004E64;

color: #FFFFFF;

border-right: 2px solid #25A18E;

}

.stSelectbox {

background-color: Orange;

color: #fb6107;

border-radius: 5px;

margin-bottom: 10px;

}

.stSelectbox label {

color: #1f2421;

font-weight: bold;

}

.stButton button:hover {

background-color: #25A18E;

transform: scale(1.05);

}

.stDataFrame, .stTable {

background-color: #E5FFF6;

color: #004E64;

border: 3px solid #004E64;

border-radius:4px;

margin-top: 20px;

}

.stSlider .stSliderLabel {

display: none !important;

}

</style>

""",

unsafe\_allow\_html=True

)

## **What is inside the sidebar:**

As per the flow I have decided to keep the selection of the Bus state name ,Bus route name and the operator type in the left side as a sidebar so I have put everything i want inside the sidebar and the logic for selecting all the things in the select box is same first we need to filter the what we want in a sql query and put it into the select box.

Here I have additionally added a Both in selecting the operator type because without this it defaultly shows the any one operator with make a table limited that’s why I added this.

with st.sidebar:

mycursor.execute(f"SELECT DISTINCT Bus\_state\_name FROM bus\_data")

State\_transport = [state[0] for state in mycursor.fetchall()]

State\_transport = st.selectbox("Select State Transport", State\_transport)

if State\_transport:

mycursor.execute(f"SELECT DISTINCT Bus\_route\_name FROM bus\_data WHERE Bus\_state\_name = '{State\_transport}'")

Bus\_route = [route[0] for route in mycursor.fetchall()]

Bus\_route = st.selectbox("Select Bus route", Bus\_route)

if Bus\_route:

mycursor.execute(f"SELECT DISTINCT Bus\_Operator\_type FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}'")

Select\_operator\_Type = [operator\_type[0] for operator\_type in mycursor.fetchall()]

Select\_operator\_Type.insert(0, "Both")

Select\_operator\_Type = st.selectbox("Bus Operator Types", Select\_operator\_Type, index=0)

## 

## Main content and logic for choosing it:

After chosen the things in the sidebar I have given 5 different option 2 select box and 3 slider to filter out the buses based on our requirement here also I have added a default in selecting bus name and bus type for the same reason i have given before by default all three in the default conditions and shows all the rows in that table for slider we need to give a three input minimum value, maximum value, and the step value.

if State\_transport and Bus\_route and Select\_operator\_Type:

mycursor.execute(f"SELECT DISTINCT BusName FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Name = [column[0] for column in mycursor.fetchall()]

Name.insert(0, "Default") # Add "Default" option to select all

Select\_operator\_Name = st.selectbox("Bus Name", Name, key="bus\_name", index=0)

# Fetch bus types based on selected operator type

mycursor.execute(f"SELECT DISTINCT BusType FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Type = [column[0] for column in mycursor.fetchall()]

Type.insert(0, "Default") # Add "Default" option to select all

Select\_Bus\_Type = st.selectbox("Bus Comfort Type", Type, key="bus\_type", index=0)

# Fetch star ratings

mycursor.execute(f"SELECT Star\_rating FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Star\_Rating = st.slider("Ratings", min\_value=0.0, max\_value=5.0, step=0.1, value=(0.0, 5.0), key="star\_rating")

# Fetch price range

mycursor.execute(f"SELECT Price FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Price = st.slider("Price Range", min\_value=0, max\_value=5000, step=50, value=(0, 5000), key="price")

# Fetch seats available

mycursor.execute(f"SELECT Seats\_available FROM bus\_data WHERE Bus\_route\_name = '{Bus\_route}' AND Bus\_Operator\_type = '{Select\_operator\_Type}'")

Seats\_available = st.slider("Seats Available", min\_value=0, max\_value=60, step=3, value=(0, 60), key="seats\_available")

## 

## **Logic behind the selection:**

By default the app will show the entire rows but if something is selected means that query is appended inside the query conditions and reflected in the table as per our selection or sliding those conditions will work only the default and both are not selected and finally the resulted query is in the mycursor and i have formatted a reaching time and departing time because it is converting into a hours that's why I used formatting and i have also fetched a column names to display it.

query\_conditions = []

if Select\_operator\_Name != "Default":

query\_conditions.append(f"BusName = '{Select\_operator\_Name}'")

if Select\_Bus\_Type != "Default":

query\_conditions.append(f"BusType = '{Select\_Bus\_Type}'")

query\_conditions.append(f"Star\_rating BETWEEN {Star\_Rating[0]} AND {Star\_Rating[1]}")

query\_conditions.append(f"Price BETWEEN {Price[0]} AND {Price[1]}")

query\_conditions.append(f"Seats\_available BETWEEN {Seats\_available[0]} AND {Seats\_available[1]}")

if Select\_operator\_Type != "Both":

query\_conditions.append(f"Bus\_Operator\_type = '{Select\_operator\_Type}'")

query = f"""

SELECT \*, DATE\_FORMAT(Departing\_Time, '%H:%i:%s') AS Departing\_Time\_, DATE\_FORMAT(Reaching\_Time, '%H:%i:%s') AS Reaching\_Time\_

FROM bus\_data

WHERE Bus\_route\_name = '{Bus\_route}' AND {' AND '.join(query\_conditions)}

"""

mycursor.execute(query)

data = mycursor.fetchall()

columns = mycursor.column\_names

## **Final pandas integration for displaying the resulted table:**

And finally I have inserted the resulting data into a pandas dataframe and I have dropped an old departing and reaching time column and I have assigned an id as an index column so the result will be shown as I expected.

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

# Drop old columns

df.drop(['Departing\_Time', 'Reaching\_Time'], axis=1, inplace=True)

# Set 'id' column as the index

df.set\_index('id', inplace=True)

# Display data without index

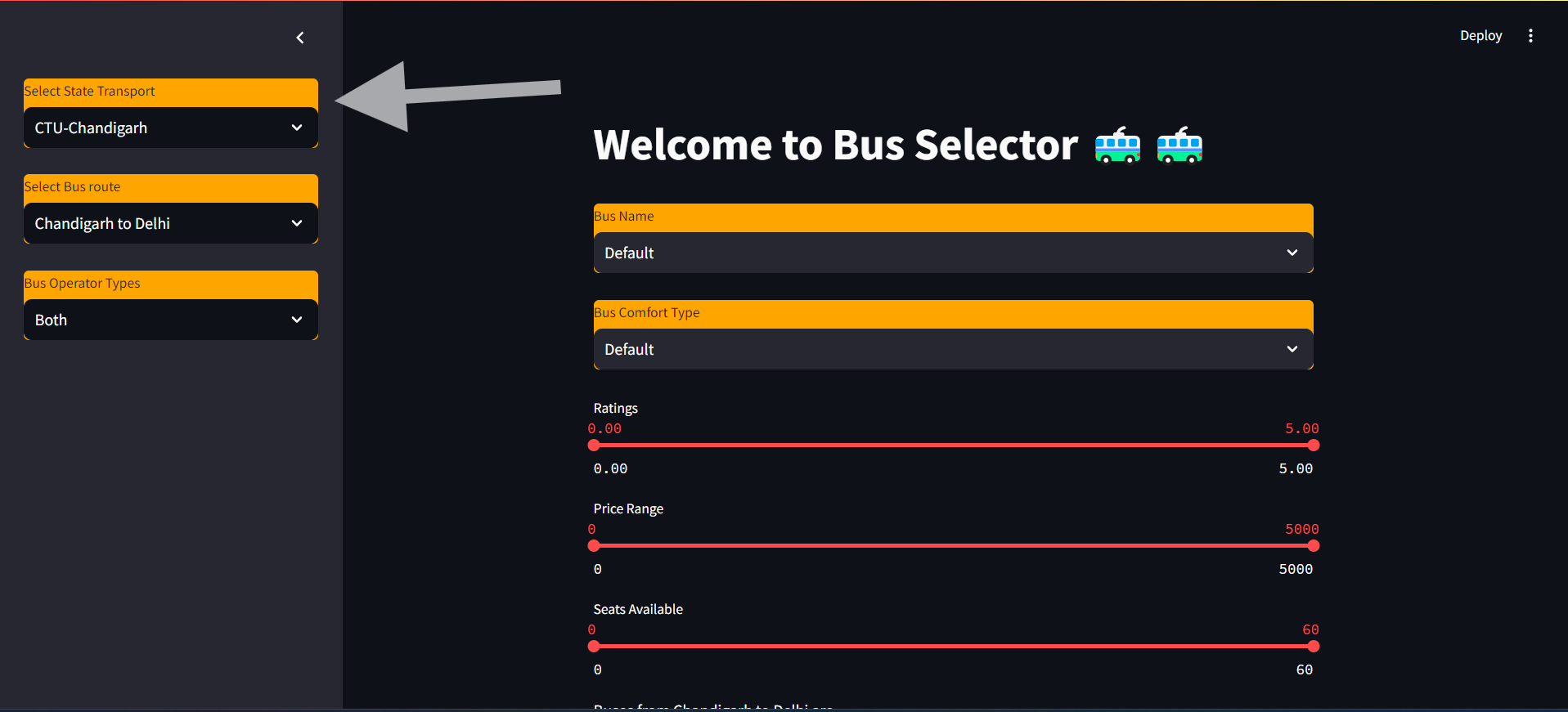
st.write(f"Buses from {selected\_table} are")

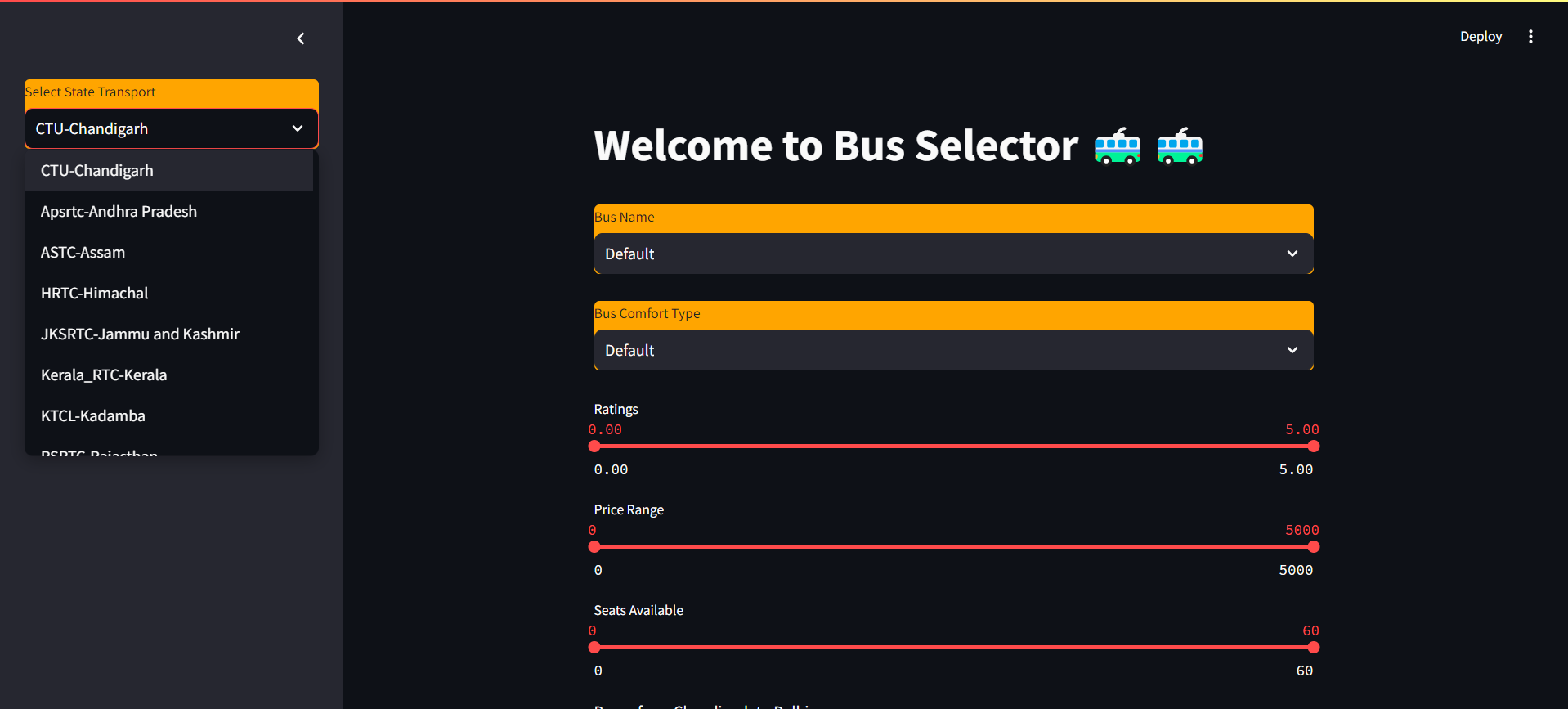
st.dataframe(df)

# **App using demo:**

## **Filtering the buses based on the State:**

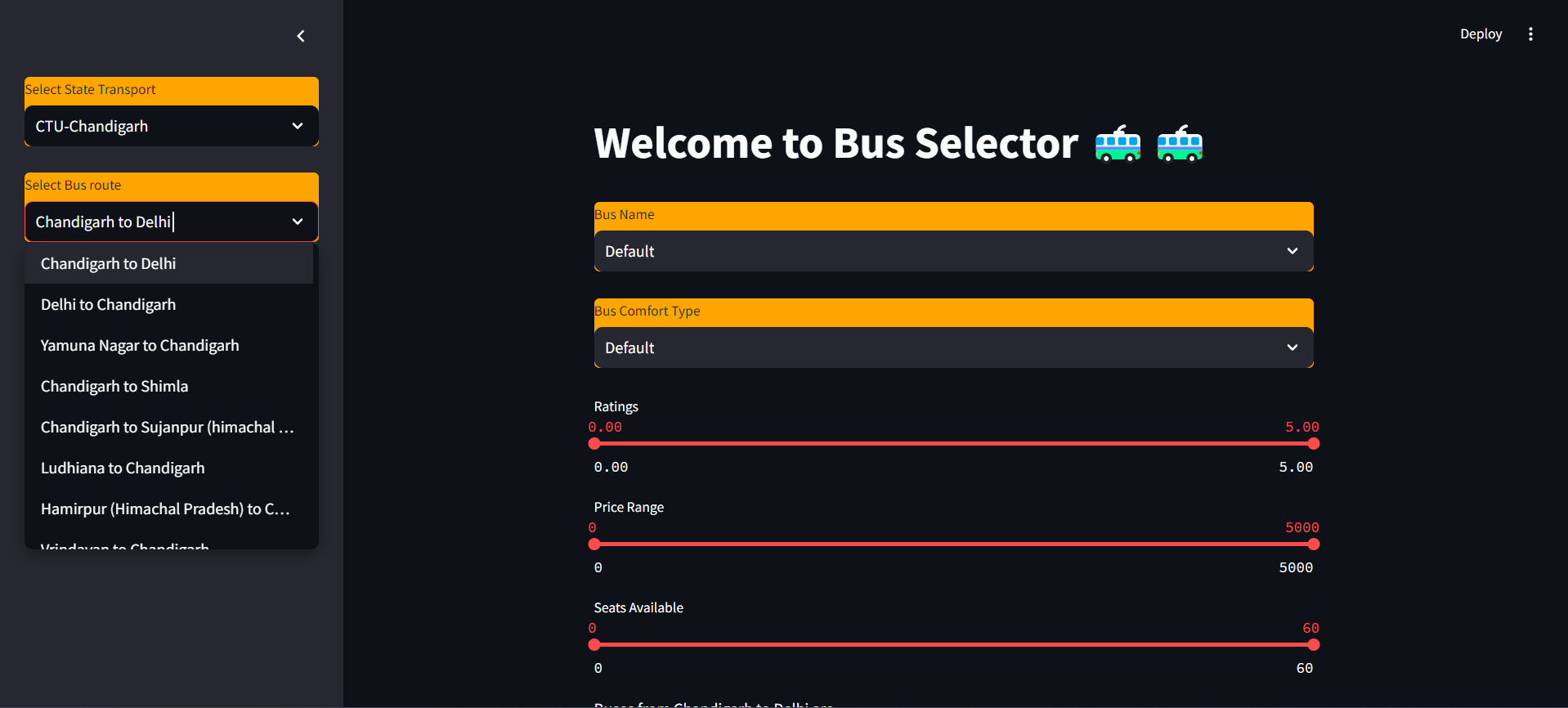
For selecting the state bus that you needed you can click the state bus transport and the dropdown box will appear then you can select a required state transport.





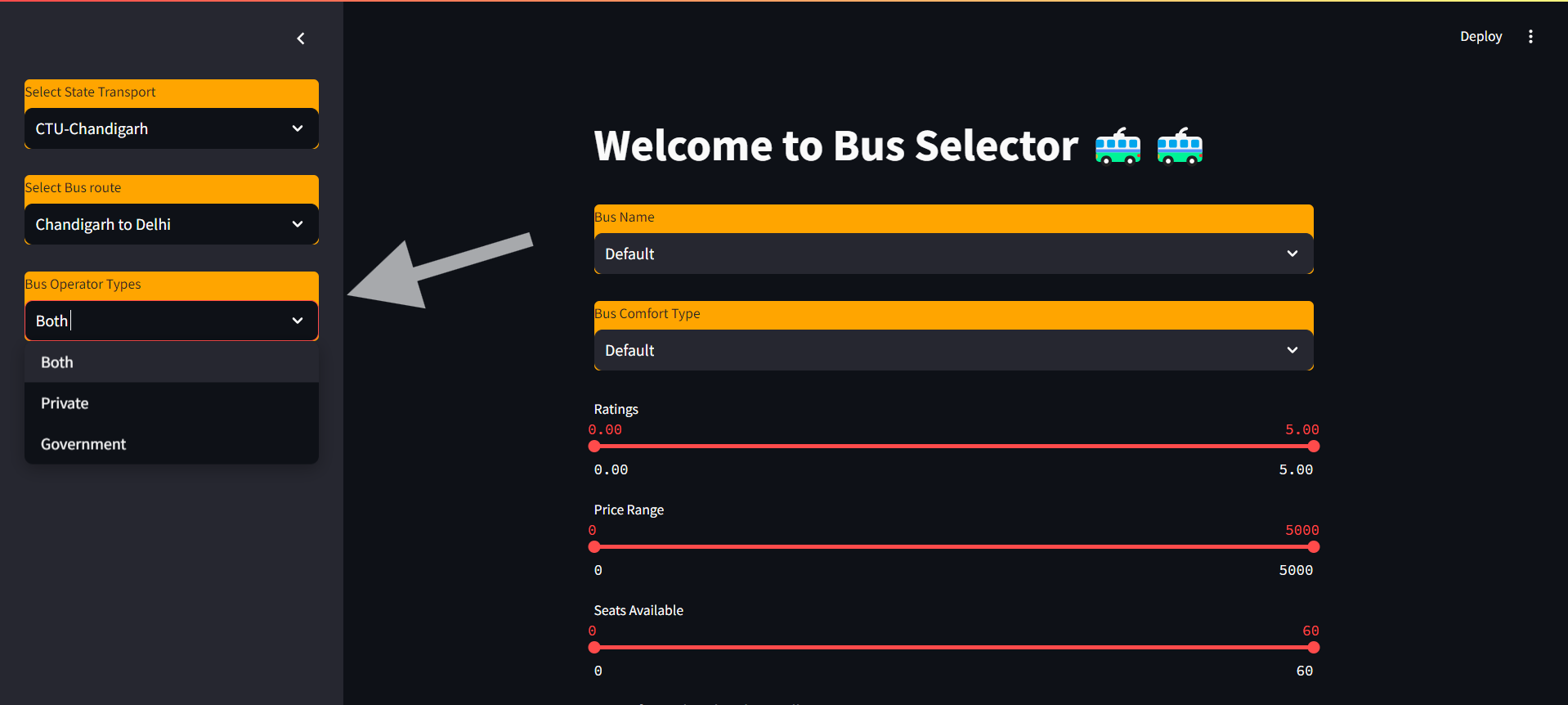
## **Filtering the buses based on the Route:**

For selecting the Route that you need you can click the Select Route and the dropdown box will appear then you can select a required Route.



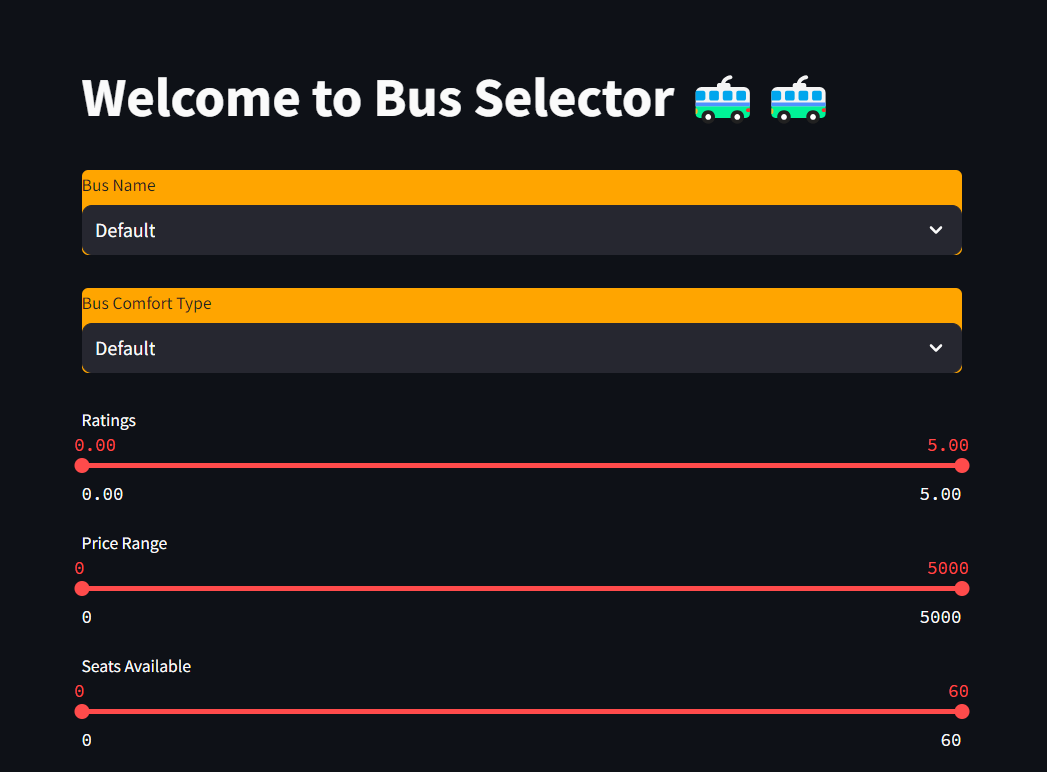
## **Filtering the buses based on the Service provider:**

For selecting the service provider that you need you can click the Bus operator type and the dropdown box will appear then you can select a service provider.



## **Filters in main page:**

There are five filters for filtering out the buses as per your needs you can filter it using bus type, bus name , rating , price and seat available.



## **Final table look:**

Finally the table looks like this.The results are completely based on the filter you have applied.



## 

## **Full final demo video:**

[Redbus\_App\_Demo](https://drive.google.com/file/d/16_Al-RhxDSwUoZ-YE4uqOimo7AgGOrbu/view?usp=sharing)