**Redbus Data Scraping with Selenium & Dynamic Filtering using Streamlit**

**This project involves three main components:**

Data Scraping with Selenium: Scraping bus route data from RedBus, including bus details such as route name, bus type, departure time, price, and seat availability.

Storing Data in SQL: Storing the scraped bus data into a MySQL database for easy retrieval and manipulation.

Creating Filters with Streamlit: Using Streamlit to create an interactive web interface, allowing users to filter and view the bus data based on various attributes like bus type, price, and star ratings.

**Project Workflow:**

1.First install the needed modules use pip and import package.

2. modules: web driver module from selenium which is used to initiate web driver for web automation.

3.By is used to locate elements on a webpage using various methods such as Id,css selectors ,xpath etc.

4.web driver wait is used for implementing explicit waits in the script.

5.Expected condition provides common conditions for waiting such as waiting for an element to be visible, clickable.

6.keys allows the script to stimulate keyboard actions.

7. Action chains enables the script to stimulate more advanced interactions like mouse movements.

8.Pandas is a powerful library for data manipulation and analysis.

9.numpy is a package for scientific computation it prefers calculation.

10.import time: time.sleep() is often used to introduce pauses between actions.

**1. Web Scraping:**

1.Initializes a new chrome browser instance using selenium to interact with the website.

2.opens specified url in the chrome browser, then driver will wait for a maximum of 10 seconds for certain elements to appear or become clickable.

3.maximize the browser window to ensure that all elements are visible.

4. Initialize empty lists to store the route names and links, define the function scrap\_page() to locate all the elements on the page that have the class route and returns the list of matching elements (route names and route links) ,for loop is used to iterate through all the elements found in bus route and extract route names and links.

5.try-except block is used to catch any error that might occur during the process. If there is an issue the error will printed the loop will continue to the next route.

6. pagination handling : Redbus website has more than one page so navigate through all pages and scroll to load all routes before scraping, here the x path identifies the specific location of the pagination container on the page then locates the next page button within the container and click the button.

7. Action chains used to stimulate scrolling to the next page button ensuring its visible and in view for clicking.

8.scraping bus item: The bus item class used to identify the bus data container on the page and scraping details for each bus.

9.Error handling: Each try-except block ensures that errors in specific operations such as finding elements or clicking buttons don’t stop the script from continuing. If error occur while scraping the bus data on error message is logged and the script moves to the next page.

10. Closing the web driver after the scraping process complete.

11.convert the scrapped data to data frame then clean it and push into my Sql.

**2. SQL Database Integration:**

Connect to the MySQL server running locally (localhost) using pymysql. connect ()

provide necessary credentials such as host, database, user, and password.

**Creating the Database and Table:**

The script first checks if the redbus database exists; if not, it creates it.

Next, the script creates a table bus\_routes1 in the redbus database with columns to store bus route details, including:

|  |  |
| --- | --- |
| **Column names** | **Data types** |
| Id | INT PRIMARY KEY AUTO INCREMENT |
| Route Name | TEXT |
| Route Link | TEXT |
| Bus Name | TEXT |
| Bus Type | TEXT |
| Departing Time | VARCHAR (100) |
| Duration | TEXT |
| Reaching Time | VARCHAR (100) |
| Star Rating | FLOAT |
| Price | FLOAT |
| Seat availability | INT |

**Inserting Data into the Table:**

After establishing the connection and ensuring the table is created, the script inserts data into the table.

The data is inserted in batches using the INSERT INTO SQL statement, and each row of data is extracted from a Pandas DataFrame (df), which contains the bus route information.

Committing the Data:

After each insert operation, the transaction is committed using myconnection.commit(). This ensures that the data is saved in the MySQL database.

**3. Streamlit App Development:**

**Database Setup:**

The MySQL database contains a table bus\_routes1, which holds data about various bus routes, including route name, bus name, type, departure time, price, star rating, seat availability, etc.

Filters and User Input:

Streamlit provides a simple user interface (UI) where users can select various filters such as:

State: A dropdown to select the state.

Route: A dropdown to select the route.

Bus Type: A dropdown to select the bus type.

Star Rating: A slider to select the minimum star rating (from 1 to

Time: A time input to select the departure time.

Price Range: A slider to filter buses by their price.

Seat Availability: A number input to filter based on the number of available seats.

**SQL Query Construction:**

The code constructs a dynamic SQL query based on the filters selected by the user. The SQL query retrieves bus routes from the MySQL database that match all selected filters.

The filters are applied to fields like state, route name, bus type, star rating, departing time, price, and seat availability.

**Fetching and Displaying Results:**

The filtered bus routes are fetched from the database and displayed in a Pandas Data Frame within the Streamlit app.

If no results match the selected filters, the message "No buses match the selected criteria" is displayed.

**Time Conversion:**

The departing time and reaching time fields are converted from time delta format to a more readable string format.