Overview and Introduction

Title:

Busly

Every Route. Every Stop. One App.

Brief Introduction:

"Busly is an innovative bus travel information platform that simplifies the process of finding, filtering, and booking bus services. From scraping routes to dynamic filtering and booking the app."

Features Summary:

- ✓ Scrape bus routes and details from a website.
- ✓ Store and organize data using SQL databases.
- ✓ Display bus information dynamically with user-friendly filters.
- ✓ Enable seat booking with real-time updates.

Tech Stack:

The primary tools;

- ✓ Python,
- ✓ Selenium,
- ✓ Pandas,
- ✓ SQL (PhpMyAdmin),
- ✓ Streamlit.

Key Features Implemented:

Database Connection:

- Establishes a MySQL connection for fetching and updating bus data.

Filter System:

- Includes robust filters for routes, bus types, ratings, fares, boarding points, dropping points, and departure times.
- Dynamically adapts to selected values.

User-Friendly Sidebar:

- Organizes filters neatly with expanders for route and bus-specific filters.

Interactive Booking Flow:

- Users can view available buses, filter results, and book seats.
- Booking reduces seat availability and confirms booking in real-time.

UI Elements:

- Displays bus details in a styled dataframe.
- Includes visual elements like images for branding and error handling.

Bus Links and Details Scraper

bus_data_automation.py

What This Script Does

This script helps scrape bus route links and detailed information about buses from a website. It uses Python and Selenium to get the data and saves it in an Excel or CSV file. The script has two main parts:

Red_bus_scraper.py

1. **`Bus_Links_Scraper`**: Finds bus route names and links.
2. **`BusDetails`**: Gets detailed information about buses, like operator name, departure time, and fare.

Key_for_scrape.py

1. **`links_scraper:`**
   - Finds and collects bus route links from the **Redbus** website.
   - Removes duplicates and saves the cleaned data in a file called ***`cleanedBD.xlsx`***.
   - This step is for getting a list of all bus routes.
2. **`route_data_scraper:`**
   - Uses the bus route links from **`cleanedBD.xlsx`**.
   - Collects detailed information about buses, such as:
     - Operator name
     - Bus type
     - Departure time
     - Ratings
     - Fare and available seats
   - Saves the details in **`bus_details.csv`**.
3. **`Choice:`**
   - When you run the script, it asks:

- Press 1 to scrape bus route links (use **`links_scraper()`**).
- Press 2 to scrape detailed bus info (use **`route_data_scraper()`**).

- Depending on what you select, the script will scrape and save the data for you.

Files Used

- **`scraping_log.txt`**: Keeps track of progress and errors.
- **`bus_data.xlsx`** / **`bus_details.csv`**: Saves the collected data.

Main Parts of the Script

Red_bus_scraper.py

1. Bus_Links_Scraper (For Route Links)

This part collects links to bus routes and their names.

What It Has:

- **`__init__(links)`**: Prepares the scraper with a list of URLs.
- **`scrape_routes(link)`**: Finds routes and links from one URL. Stops after 5 pages or if no more pages are found.
- **`scrape_all()`**: Goes through all the links you give it.
- **`save_results(filename='bus_data.xlsx')`**: Saves the routes and links to an Excel file.

Example Use:

```
links = ["https://www.redbus.in/online-booking/astc",]
scraper = Bus_Links_Scraper(links)
scraper.scrape_all()
scraper.save_results()
```

2. BusDetails (For Bus Info)

This part collects detailed information about the buses on each route.

What It Has:

- **`__init__(route_links)`**: Prepares the scraper with a list of route links.
- **`scrape_route_details()`**: Gets all the details about buses on each route, including:
  - Operator name
  - Bus type
  - Departure and arrival times
  - Boarding and dropping points

- Fare and available seats
- Ratings

- **`_load_all_buses()`**: Makes sure all buses on a page are loaded by scrolling and clicking buttons.
- **`save_results(filename='bus_details.csv')`**: Saves the bus details in a CSV file.

Example Use:

```
route_links = ["https://example.com/route-detail1",
"https://example.com/route-detail2"]
detail_scraper = BusDetails(route_links)
detail_scraper.scrape_route_details()
detail_scraper.save_results()
```

Key_for_scrape.py

Main Script *(User Choice)*

The main script allows the user to choose between scraping bus route links or scraping detailed bus information. The flow is as follows:

Code Example:

```
from Red_bus_scraper import Bus_links_scraper, BusDetails
import pandas as pd

def links_scraper():
    links_to_scrape = [
        'https://www.redbus.in/online-
booking/apsrtc/?utm_source=rtchometile',...    ]

    scraper = Bus_links_scraper(links_to_scrape)
    scraper.scrape_all()
    scraper.save_results()

    df = pd.read_excel('bus_data.xlsx')
    cleanedBD = df.drop_duplicates()
    cleanedBD.to_excel('cleanedBD.xlsx', index=False)
    print("Data cleaned and saved to cleanedBD.xlsx")

def route_data_scraper():
    xlsx_path = r'C:\Users\Windows
11\OneDrive\Desktop\RedBus\cleanedBD.xlsx'
    scraped_links = pd.read_excel(xlsx_path)
    route_links = scraped_links['route_link'].tolist()

    scraper = BusDetails(route_links)
    scraper.scrape_route_details()
    scraper.save_results()
```

Choice Code:

```
choice = int(input("Enter your choice (for scrape link [1] &
for scrape data [2]): "))

if choice == 1:
    links_scraper()
elif choice == 2:
    route_data_scraper()
else:
    print("Invalid choice. Please enter 1 or 2.")
```

How to Run the Script

1. **Choose an Option**:
    a. Enter **1** to scrape bus route links and remove duplicate then save them in **`cleanedBD.xlsx`**.
    b. Enter 2 to scrape detailed bus information from the links in **`cleanedBD.xlsx`**.
2. **Output**:
    a. Route links are saved in **`cleanedBD.xlsx.`**
    b. Bus details are saved in **`bus_details.csv`**.

Error Handling

- **Page Loading**: Waits until the page elements are ready.
- **Logs**: Saves any issues or progress in **`scraping_log.txt`**.
- **Skipping Errors**: If something fails, it moves to the next item and logs the error.

Where the Data Goes

1. **Bus Routes**:
    `a.` Saved as **`bus_data.xlsx.`**
    b. Includes columns for **`route_name`** and **`route_link.`**
2. **Bus Details**:
    a. Saved as **`bus_details.csv.`**
    b. Includes columns like `Route Name`, `Operator`, `Departure Time, Ratings, Fare`, etc.

SQL

bus_data_to_sql3.py

What this Script Does

This script interacts with a **PhpMyAdmin** database to import bus data from a CSV file and store it in a structured format. Here's how it works:

Class: *Data_base*

The main part of the script.

1. Initialization *(__init__)*

- Path to the CSV file.
- **Actions:** Connects to the **PhpMyAdmin** database (***bus_data***), creates it if it doesn't exist, and ensures the table is there.

2. **Table Creation *(create_table)***

- **Table Name: *buses***
- **Columns:**
  - ***id*:** Auto-incremented ID.
  - ***route_name, route_link, bus_name, etc*.:** Fields for bus details.
- Unique Constraint: Ensures no duplicate entries for the same bus.

3. **Duplicate Check *(is_row_existing)***

This checks if a bus with the same ***route_link***, ***bus_name***, and ***departing_time*** already exists in the database.

4. **Bus Type Categorization *(categorize_bus_type)***

Purpose: Standardizes bus types like S**eater, Sleeper, AC, etc**., based on the CSV data.

5. **Data Insertion *(insert_data_from_csv)***

- Path to the CSV file.
- **Actions:** Reads and processes the CSV, cleans up data, and inserts it into the buses table. It skips duplicates.

6. **Close Connection *(close_connection)***

Closes the connection to the database once everything is done.

It handles the database connection, creates the necessary table, and inserts data from the CSV.

How to Use the Script:

Set Up PhpMyAdmin:

a. Make sure **XAMPP** is installed and **PhpMyAdmin** is running locally.

b. Ensure the **root** user can access the database, either with no **password** or with the correct one.

CSV File:

c. The script expects the CSV file to have these columns: ***route_name***, ***route_link***, ***bus_name***, ***bus_type***, ***departing_time***, ***etc.***

Running the Script:

d. The script will connect to **PhpMyAdmin**, create the ***bus_data*** database, and insert data from the CSV.

e. Duplicates will be skipped.

After Running:

f. The ***bus_data*** database will have a buses table filled with ***bus details.***

SQL Table Example

```
CREATE DATABASE IF NOT EXISTS bus_data;


USE bus_data;

CREATE TABLE IF NOT EXISTS buses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    route_name TEXT,
    route_link TEXT,
    bus_name TEXT,
    bus_type TEXT,
    departing_time TIME,
    boarding_point TEXT,
    duration TEXT,
    reaching_time TIME,
    dropping_point TEXT,
    star_rating FLOAT,
    fare FLOAT,
    seats_available INT,
    UNIQUE(route_link, bus_name, departing_time)
```

```
);
```

This SQL sets up the structure for the bus data with proper columns and a unique constraint to prevent duplicates.

Streamlit

bus_booking_app4.py

Script Overview:

This Streamlit app (***bus_booking_app4.py***) allows users to interact with a ***MySQL*** database to search and book buses based on their preferences. The app provides dynamic filtering options, booking, and updates bus seat availability in real time. Below is a summary of the main sections:

Main Features:

1. **Filters**:
    a. **Route Filters**: Allows the user to select routes by "**From**" and "**To**" locations.
    b. **Bus Filters**: Filters buses by bus type (with the ability to include custom types) and by **rating** and **fare range**.
    c. **Boarding/Dropping Points and Departure Time Filters**: Further refines results based on user selection for boarding point, dropping point, and departure time.
2. **Booking System**:
    a. **Available Buses**: Displays buses based on the selected filters.
    b. **Seat Availability**: The app reduces available seats in real-time when a booking is confirmed.
    c. **Confirmation**: Displays booking details and a confirmation message after a successful booking.
3. **Database Interaction**:
    a. Fetches distinct values for filtering, and then executes SQL queries to filter the buses based on user-selected criteria.
    b. Updates the MySQL database to decrease the number of seats available after booking.
4. **UI Customizations**:
    a. **Hidden Streamlit UI Elements**: Using custom CSS to hide unnecessary UI components.
    b. **Error Image**: Displays an error image if no buses match the selected criteria.

Key Components:

1. **Database Connection**:
    a. The app connects to a MySQL database (***`bus_data`***) to fetch and update bus information.
2. **Dynamic Filters**:
    a. ***`distinct_filters:`*** Retrieves unique values for filtering options (routes, bus types).
    b. ***`fetch_filters`***: Fetches filtered bus data from the database based on user input.
3. **Filter and Display**:
    a. ***`filter_bd_point:`*** Filters based on boarding point, dropping point, and departure time.
    b. ***`booking_data:`*** Displays the filtered bus data, with a "**Book Now**" button to confirm bookings.
4. **Booking Confirmation**:
    a. ***`book_bus:`*** Updates the bus's seat availability in the database.
    b. ***`dp_Bus_Details:`*** Displays the details of the selected bus for user confirmation.

How to Run the Script:

1. **Setup**:
    a. Ensure that the MySQL database (***`bus_data`***) exists and contains the appropriate bus information in the `buses` table.
    b. Install the required Python libraries: ***`pip install streamlit`***, ***`mysql-connector`***, ***`pandas`***

2. **Images**:
    a. Add an image file named ***`SelecteCriteria.png`*** to the working directory to show an error when no buses match the criteria.
3. **Running the App**:
    a. Save the script as ***`bus_booking_app4.py`***.
    b. In the terminal, run: **streamlit run bus_booking_app4.py**

Additional Considerations:

- **Database Population**: If the ***`buses`*** table is not yet populated, use the CSV import script provided earlier to insert data into the table.

- **Error Handling**: The script includes error handling to manage database connection issues, seat availability, and booking errors.

Summary:

This script creates an interactive bus booking application where users can filter buses by route, type, and fare, then book available buses with real-time seat updates. The custom UI elements and hidden Streamlit components improve the overall user experience, making it efficient and clean.