Introduction

Project Overview :

This project implements a Travel Booking System named "Logic Express," a bus travel agency. It's a console-based application written in C++ that allows users to manage user details and bus ticket bookings. The system handles user registration, booking new tickets, modifying existing bookings, canceling bookings, and viewing available seats and booking information.

Key Features :

- **User Management** : Add new users with unique IDs, names, mobile numbers, and cities.
- **Booking Management** : Book bus tickets, including passenger name, source, destination, seat number, date, and times.
- **Unique Ids** : Ensures unique User IDs and Booking IDs.
- **Seat Allocation** : Manages seat availability and assigns unique seat numbers for bookings.
- **Booking Modification** : Allows users to change the destination which automatically changes the departure time, arrival time, and price of an existing booking.
- **Booking Cancellation** : Enables cancellation of bookings and makes the seat available again.
- **Search Functionality** : Search for bookings by passenger name or destination.
- **View Options** : Display all registered users and all existing bookings.
- **Seat Availability Display** : Shows which seats are booked and which are free.
- **Input Validation** : Basic validation for user inputs like names, mobile numbers, and seat choices.

Enhancement

- **Robust Input Validation and Error Handling :**
    1. **Numeric Input Validation :** Implement a robust input loop for all numeric inputs (UserID, MobileNo, SeatNo, BookingID) to handle non-numeric input gracefully and prevent infinite loops. Currently, entering characters for cin >> int_variable can lead to cin failing and entering an infinite loop without clearing the error state and ignoring remaining input.
    2. **String Input Handling :** Improve getline(cin, string_variable) usage to ensure it correctly captures the entire line after numeric inputs, preventing issues where the newline character from cin >> causes getline to read an empty string. The current cin.clear() and cin.ignore() are a good start but ensure they are consistently applied where needed.
    3. **Specific Error Messages :** Provide more descriptive and user-friendly error messages for all invalid inputs.
    4. **Database Interaction Error Handling :** Although you're using text files, the concept of handling "database interaction" errors applies. Implement checks for successful file openings (.is_open()) and handle cases where files might be corrupted or inaccessible.

- **Bus and Route Management :**
    1. Multiple Buses and Routes : Currently, the system assumes a single bus (25 seats) and a fixed boarding point (“Jodhpur”) with only a few fixed destinations. Expand to manage:
    2. Multiple Buses : Define different buses with varying capacities.
    3. Multiple Routes : Allow different routes with unique sources, destinations, and schedules.
    4. Dynamic Departure/Arrival Times : Instead of fixed times, allow for flexible scheduling of trips.

5. Trip Management : Define specific trips with a chosen bus, route, date, and times.

- **Advanced Booking Features :**
    1. **Seat Selection Interface :** With a GUI, provide a visual representation of the bus layout where users can click to select seats.
    2. **Ticket Generation/Confirmation :** Generate a simple text-based or PDF ticket/confirmation with all booking details.
    3. **Refund Policy :** Implement logic for refunds in case of cancellation, possibly with cancellation fees.

- **Sorting and Filtering :**
    1. **Display Sorting :** Add options to sort the lists of users and bookings (e.g., sort bookings by date, price, or passenger name; sort users by ID or name).
    2. **Advanced Search :** Implement more flexible search options, such as searching bookings by date range, user ID, or a combination of criteria.

- **Backup and Restore :**
    1. **Data Backup :** Develop a mechanism to create backups of the data files (.txt files or database if implemented) and a way to restore them.

Limitations

- **Input Validation Gaps Leading to Runtime Issues/Infinite Loops :**
    1. **Mobile Number Input :** As noted, entering non-numeric characters for MobileNo causes cin to fail, leading to an infinite loop in the do-while loop within AddUser() if the input buffer is not properly cleared and ignored. This needs more robust error state checking and input discarding.
    2. **Username/City Spaces :** The isWhiteSp() function correctly flags strings with spaces as invalid. However, for names like "John Doe" or cities like "New Delhi," this is a practical limitation. The getline is used, but the isWhiteSp check then prevents such inputs. The valid_NameFormat also prevents spaces. This needs to be clarified if spaces are truly disallowed for a single-word name, or if the intention is to allow multi-word names without special characters.
    3. **Partial getline Input Verification :** If a user enters "John Doe" for a username, getline reads it all, but isWhiteSp rejects it. If the intention is to read only "John", the current approach with isWhiteSp is consistent, but if multi-word names are desired, the validation logic needs to change.

- **Single Bus and Fixed Route/Boarding Point :**
    1. **Fixed Capacity :** The system is hardcoded for a single bus with a fixed capacity of 25 seats. There's no mechanism to add or manage multiple buses with different capacities.
    2. **Single Route/Boarding :** All bookings originate from a fixed "Jodhpur" boarding point and can only go to the four predefined destinations. There's no flexibility to add new routes, different source locations, or manage multiple trips.

- **Limited Data Persistence :**
    1. **Flat File Storage :** Data is stored in simple text files (.txt), which is prone to issues like:
        a) **Data Corruption :** Easily corrupted if the program crashes or if files are manually altered incorrectly.
        b) **Concurrency Issues :** Not suitable for multi-user access (though the current program is single-user).
        c) **Querying Limitations :** Searching and filtering data is inefficient as it requires reading the entire file every time.
        d) **Scalability Concerns :** Performance degrades as the number of records increases.

- **No User Authentication or Roles :**
    1. **Lack of Security :** Any user can access, add, edit, or delete any booking or user record by simply knowing or guessing their respective Ids. There are no login credentials or access controls.
    2. **No Admin Features :** No separate roles or privileges for system administrators to manage the bus operations (e.g., adding/removing destinations, changing prices, managing bus availability).

- **Basic Search and Display Functionality :**
    1. **Limited Search Criteria :** Bookings can only be searched by passenger name or destination. There are no options to search by booking ID, user ID, date, or other fields.
    2. **No Sorting/Filtering :** The displayed lists of users and bookings cannot be sorted or filtered based on various criteria.
    3. **Simple Seat Display :** The view_available_seats() function shows a 5x5 grid, but display_available_seats() just lists booked/free seats in a linear fashion, which is less intuitive.

- **Hardcoded Values :**
    1. **Departure Time :** The departure time is fixed at "02:00" for all bookings.
    2. **Price Allocation :** Prices are directly tied to the destination choice via a switch-case, lacking flexibility for dynamic pricing or fare classes.
    3. **Date Logic :** Only allows booking for "tomorrow's date," which is highly restrictive for a real-world booking system.

- **Code Structure and Maintainability :**
    1. **Global Variables/Functions :** Many functions operate directly on files without being encapsulated within classes that manage file operations, which can make the code harder to extend and test.
    2. **Repetitive File Operations :** Opening, reading, writing, closing, removing, and renaming temporary files are repeated across multiple functions (update_seats, EditBooking, CancelBooking), which could be refactored into helper functions or methods.
    3. **Limited Comments :** While some comments exist, more detailed explanations for complex logic blocks or design decisions would improve long-term maintainability.

Abstract

Modules imported :

1. **<iostream>** : For standard input/output operations (e.g., cin, cout).
2. **<string>** : For string manipulation.
3. **<fstream>** : For file input/output operations (e.g., ifstream, ofstream).
4. **<vector>** : To use dynamic arrays (e.g., for storing booked and free seats).
5. **<cctype>** : For character handling functions (e.g., toupper, isspace, isalpha).
6. **<cstdlib>** : For general utilities, including rand() and srand() for random number generation.
7. **<ctime>** : For time-related functions (e.g., time, localtime, mktime) to handle dates.
8. **<limits> :** To use numeric_limits for clearing input buffer.

Files used :

a) **Permanent Files :**

- i. **Seats.txt :** Stores the current status of each seat (either its number if free, or "0" if booked).
- ii. **UserDetails.txt :** Stores user information (UserID, UserName, MobileNo, City).
- iii. **bookingDetails.txt :** Stores booking information (BookingID, UserID, PassengerName, Source, Destination, SeatNo, Date, DepartureTime, ArrivalTime, Price).

b) **Temporary Files :**

- i. **w.txt :** Used during seat updates.
- ii. **tempBooking.txt :** Used during booking edits and cancellations.
- iii. **p.txt :** Used during seat updates when a booking is canceled.

Functions used :

1. **cap(string &s) :** Converts a given string to uppercase.
2. **isEmp(string s) :** Checks if a string is empty and prints a message if it is.
3. **isWhiteSp(string s)** : Checks if a string contains any whitespace characters and prints a message if it does.
4. **Date_Tomorrow()** : Returns tomorrow's date in "DD-MM-YYYY" format.
5. **file_exists()** : Checks if Seats.txt exists.
6. **view_available_seats()** : Displays the current state of seats from Seats.txt, showing available seat numbers or blanks for booked seats.
7. **zero_seats()** : Checks if all 25 seats are booked.
8. **update_seats(int dseat)** : Updates the Seats.txt file by setting a booked seat to "0".
9. **seat_available(int seat)** : Checks if a given seat number is available (not "0") in Seats.txt.
10. **seats()** : Manages the seat selection process, displays available seats, takes user input, and updates the Seats.txt file.
11. **Booking class** :
    - **Constructor** : Initializes booking details.
    - **add(ofstream &file)** : Writes a booking object's data to a file.
    - **load(ifstream &file)** : Reads booking data from a file into a booking object.
    - **print(int a, int b, int c, int d, int e, int f)** : Prints formatted booking details.
12. **User class** :
    - **Constructor** : Initializes user details.
    - **add(ofstream &writer)** : Writes a user object's data to a file.
    - **load(ifstream &reader)** : Reads user data from a file into a user object.
    - **print(int a, int b, int c)** : Prints formatted user details.
13. **is_copy_booking(int r, Booking k, ifstream &file)** : Checks for duplicate Booking IDs.
14. **is_copy_user(int r, User k, ifstream &file)** : Checks for duplicate User IDs.
15. **num_format(long int num)** : Validates if a mobile number has exactly 10 digits.
16. **valid_NameFormat(string a)** : Checks if a string contains only alphabetic characters or underscores.

17. **verify_user(int uid)** : Verifies if a User ID exists in UserDetails.txt.
18. **select_dest(int c)** : Returns the destination string based on a numerical choice.
19. **menu_dest()** : Displays the menu of destination choices.
20. **get_dest(int choice)** : Prompts the user for a destination choice and validates it.
21. **get_arrivalTime(int c)** : Returns the arrival time based on the destination choice.
22. **get_price(int c)** : Returns the ticket price based on the destination choice.
23. **AddBooking()** : Guides the user through the process of adding a new booking, including user verification, seat selection, and generating a unique booking ID.
24. **EditBooking()** : Allows a user to modify the destination, departure time, arrival time, and price of an existing booking based on its Booking ID.
25. **CancelBooking()** : Cancels a booking by its Booking ID and makes the corresponding seat available again.
26. **ViewAllBookings()** : Displays all existing bookings in a formatted table.
27. **SearchBooking()** : Searches for and displays bookings by passenger name or destination.
28. **AddUser()** : Guides the user through adding a new user, ensuring unique IDs and validating inputs.
29. **ViewAllUser()** : Displays all registered users in a formatted table.
30. **display_available_seats()** : Shows a list of booked seats and free seats.
31. **main()** : The main function that initializes the Seats.txt file if it doesn't exist and presents the main menu for the travel booking system.

How to Use

Compiling and Running the Program :

- **Save the Code** : Save the provided C++ code as a .cpp file (e.g., travel_booking.cpp).
- **Compile** : Open a terminal or command prompt and use a C++ compiler (like g++) to compile the code:
  g++ travel_booking.cpp -o travel_booking
  This command compiles the source code and creates an executable file named travel_booking (or travel_booking.exe on Windows).
- **Run** : Execute the compiled program from the terminal:
  ./travel_booking
  (On Windows, you might just type travel_booking.exe or travel_booking).

Specific Inputs

The program is interactive and will prompt you for inputs at various stages:

1. **Menu Choices :** Enter the number corresponding to the desired action (e.g., 1 for "Add user", 3 for "Add booking").
2. **User Details :** When adding a user, you'll be asked for a UserID, User Name, MobileNo, and City.
3. **UserID :** Must be a unique positive integer.
4. **UserName :** Should not be empty, contain whitespaces, or special characters (except underscore _).
5. **MobileNo :** Must be a 10-digit number.
6. **Booking Details :** When adding a booking, you'll need to provide an existing User ID. The system will then guide you through seat selection and destination choice.
7. **Seat Selection :** When booking a ticket, available seats (1-25) will be displayed. You'll need to enter an available seat number.
8. **Edit/Cancel Booking :** You'll be prompted to enter the BookingID for the booking you wish to edit or cancel.
9. **Search Booking :** Enter a Passenger Name or Destination to search for bookings. This input also has validation rules similar to User Name.

System Requirements

Minimum Hardware Requirements

- **Processor** : Any modern CPU (e.g., Intel Core i3 or equivalent).
- **RAM** : 512 MB or more (1 GB recommended for smoother operation).
- **Storage** : A few MBs of free disk space for the executable and text files.

Software Requirements

Operating System :

1. Windows (Windows 7 or later)
2. macOS (any recent version)
3. Linux (any modern distribution)

C++ compiler :

1. GCC (GNU Compiler Collection) version 5.0 or higher (recommended).
2. Clang, Microsoft Visual C++, or any other C++11 (or later) compliant compiler.
3. **Terminal/Command Prompt** : A standard command-line interface to compile and run the program.