**Final Project**

*‘Journey Guide’*



**Session: 2023 – 2027**

**Submitted by:**

Saad Tahir 2023-CS-62

Muhammad Talha 2023-CS-70

**Supervised by:**

Sir Nazeef-ul-Haq

Sir Waseem

**Course:**

CSC-200L Data Structures and Algorithms Lab

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

Contents

[**Introduction:** 4](#_Toc185244786)

[**Objectives:** 4](#_Toc185244787)

[ **Efficient Route Calculation:** 4](#_Toc185244788)

[ **Graph Representation:** 4](#_Toc185244789)

[ **Pathfinding Algorithms:** 4](#_Toc185244790)

[ **Data Management:** 4](#_Toc185244791)

[ **User Interface:** 4](#_Toc185244792)

[ **Cost Calculation:** 4](#_Toc185244793)

[ **Journey Customization:** 4](#_Toc185244794)

[ **Scalability:** 4](#_Toc185244795)

[ **Testing and Validation:** 4](#_Toc185244796)

[ **Documentation:** 4](#_Toc185244797)

[**Incentive:** 4](#_Toc185244798)

[**Execution Details:** 5](#_Toc185244799)

[1. **Graph Construction:** 5](#_Toc185244800)

[**Algorithm Implementation:** 5](#_Toc185244801)

[3. **Dynamic Updates:** 5](#_Toc185244802)

[4. **Data Storage:** 5](#_Toc185244803)

[5. **Execution Flow:** 5](#_Toc185244804)

[**DSA Implementation:** 5](#_Toc185244805)

[**1.** **Data Structures:** 5](#_Toc185244806)

[**2.** **Algorithm Implementation:** 6](#_Toc185244807)

[**Core Functionalities:** 6](#_Toc185244808)

[**Graph Construction:** 6](#_Toc185244809)

[**User Interaction:** 7](#_Toc185244810)

[**Dynamic Updates:** 7](#_Toc185244811)

[**Wireframes:** 7](#_Toc185244812)

[**1.** **Main Screen:** 7](#_Toc185244813)

[**2.** **Account Creation:** 7](#_Toc185244814)

[**3.** **After Login Screen:** 8](#_Toc185244815)

[**4.** **Predefined Cities List:** 9](#_Toc185244816)

[**5.** **Ticket Booking:** 10](#_Toc185244817)

[**6.** **Display Price:** 10](#_Toc185244818)

[**7.** **E-Ticket Generation:** 11](#_Toc185244819)

[**8.** **Ticket History of Logged-In User:** 12](#_Toc185244820)

[**Languages and Header Files Used:** 12](#_Toc185244821)

[**Languages:** 12](#_Toc185244822)

[**Header Files:** 12](#_Toc185244823)

[**Future Enhancements:** 13](#_Toc185244824)

**Table of Figures**

[Figure 1-Main Screen 7](#_Toc185244311)

[Figure 2-Sign Up Page 8](#_Toc185244312)

[Figure 3-Available Facilities 8](#_Toc185244313)

[Figure 4-List Of Cities 9](#_Toc185244314)

[Figure 5-Ticket Booking 10](#_Toc185244315)

[Figure 6-Price 10](#_Toc185244316)

[Figure 7-E-Ticket 11](#_Toc185244317)

[Figure 8-Ticket Generation 12](#_Toc185244318)

# **Introduction:**

The "Journey Planner" project is a system designed to assist users in planning their journeys between various cities by calculating the most efficient and cost-effective routes. Using advanced data structures and algorithms, it analyzes different possible paths and travel costs to provide the best options for the user. The system supports essential functionalities like route finding, fare calculation, and ticket booking. With a user-friendly interface and a set of predefined cities, the project aims to provide a seamless and effective travel planning experience. The implementation incorporates graphs, pathfinding algorithms such as Bellman-Ford and Floyd-Warshall, and file handling for persistent data storage. This project serves as a tool for anyone looking to plan their journey efficiently.

# **Objectives:**

* **Efficient Route Calculation:** To implement algorithms for calculating the most efficient and cost-effective travel routes between cities, considering factors such as distance, time, and cost.
* **Graph Representation:** To represent cities and routes as a graph data structure, where cities are nodes, and routes (with distances or costs) are edges.
* **Pathfinding Algorithms:** To apply pathfinding algorithms (such as Bellman-Ford and Floyd-Warshall) to find the shortest and most optimal routes between cities.
* **Data Management:** To implement file handling mechanisms for storing, retrieving, and updating journey details, city information, and user preferences in a persistent manner.
* **User Interface:** To design a user-friendly interface that allows users to input their journey details (such as source city, destination city, and travel preferences) and view the suggested routes.
* **Cost Calculation:** To calculate the total cost of a journey, factoring in different travel routes, means of transport, and time.
* **Journey Customization:** To allow users to customize their journey by selecting preferred routes, modes of transport, and other travel preferences.
* **Scalability:** To ensure that the system can handle a large number of cities and routes, enabling users to plan journeys across vast networks.
* **Testing and Validation:** To thoroughly test the system for correctness, efficiency, and reliability, ensuring that the journey planning process works as expected.
* **Documentation:** To provide clear documentation and user guides to assist users in navigating the journey planner tool effectively.

# **Incentive:**

The motivation behind the "Journey Planner" project stems from the growing need for efficient and optimized travel planning in an increasingly connected world. With the vast number of cities, routes, and transport options available today, planning the most efficient and cost-effective journey can be a daunting task. Traditional methods of route planning, often involving manual calculations or disjointed tools, can be time-consuming and prone to error.

This project aims to provide a streamlined, user-friendly solution for journey planning, leveraging the power of graph theory and algorithms. By representing cities and routes as graphs, we can utilize advanced pathfinding algorithms to calculate the shortest and most optimal routes, considering factors like cost, distance, and time. The motivation lies in creating a tool that not only simplifies the journey planning process but also ensures that users can make informed decisions about their travel routes, saving both time and money.

Furthermore, with the rise of digital solutions for everyday tasks, this project seeks to enhance the user experience by offering a customizable and scalable platform that adapts to individual travel preferences and provides practical solutions to real-world travel challenges.

# **Execution Details:**

The Journey Planner project is executed through the main.cpp file, which serves as the central entry point for the program. The main operations of the project are:

1. **Graph Construction:** The program initializes a graph with cities as nodes and routes (distances) as edges. This is achieved using an adjacency list representation.
2. **User Input:** The user provides the source and destination cities, which are processed to determine the best possible routes.

## **Algorithm Implementation:**

* + **Shortest Path Calculation**: The program computes the shortest path between the cities using **Floyd-Warshall algorithm** and **Bellman-Ford algorithm**, depending on the conditions.
  + **Route Display**: After computing the shortest path, the program displays the optimal route along with the total distance and travel time.

1. **Dynamic Updates:** The graph allows for dynamic modifications, where new cities and routes can be added during runtime.
2. **Data Storage:** The cities and routes are stored in a graph data structure, enabling efficient access and modifications.
3. **Execution Flow:** The main.cpp file coordinates all operations, including graph construction, user interaction, algorithm execution, and displaying results, ensuring a smooth user experience.

# **DSA Implementation:**

The project uses several data structures and algorithms to facilitate route planning and optimize flight paths between cities. Here is a breakdown of the key data structures and algorithms implemented:

## **Data Structures:**

* + **CityData Structure**:
    - The CityData struct holds an array cost[] that stores the cost (or distance) from a given city to other cities. It also holds an array[] that stores the city indices corresponding to the shortest paths.
    - It is used for representing the costs of traveling from a city to all other cities.
  + **InitialCosts Structure**:
    - The InitialCosts struct holds the original direct flight costs between cities in the cost[] array, representing the initial fare between cities before any algorithmic optimization.
  + **datamodule Class**:
    - The datamodule class initializes and manages city names and routes. It contains an array city[] with the names of 15 cities and a method costdeclaration() to initialize the cost (distance) matrix using hardcoded values in a 2D array.
  + **bellmanfordalgorithm Class**:
    - Inherits from datamodule and implements the **Bellman-Ford algorithm** and **Floyd-Warshall algorithm** for shortest path calculation. It updates the journey[] array, which tracks the shortest distances and the cities visited along the shortest path.
    - **Bellman-Ford**: A graph algorithm used for finding the shortest paths in a graph, particularly for graphs with negative weight edges. It iterates through all edges to update the distance estimates iteratively.
    - **Floyd-Warshall**: A dynamic programming algorithm used to find the shortest paths between all pairs of nodes in a graph. It updates the journey[] array for every possible path.
  + **filehandlingmodule Class**:
    - Manages user login, account creation, and ticket history.
    - **login()**: Validates user credentials by checking against a file (login.txt).
    - **createaccount()**: Creates a new account by appending user credentials to login.txt.
    - **ticket()**: Stores ticket details, including user information and passenger details, in ticket.txt.

## **Algorithm Implementation:**

* + **Shortest Path Calculation**:
    - **Bellman-Ford Algorithm**: The Bellman-Ford algorithm is used to compute the shortest path from a source city to other cities by iteratively relaxing the edges of the graph.
    - **Floyd-Warshall Algorithm**: This is another shortest path algorithm that calculates the shortest paths between all pairs of cities, updating the journey[] array.

# **Core Functionalities:**

**Graph Construction:** The graph representing cities and routes is initialized with city names and a 2D array of costs.

**User Interaction:** Users can log in, view cities, and check routes between cities, including the available flight prices, using both direct and indirect routes.

**Dynamic Updates:** The program allows users to book tickets, store ticket information, and review ticket history.

# **Wireframes:**

## **Main Screen:**

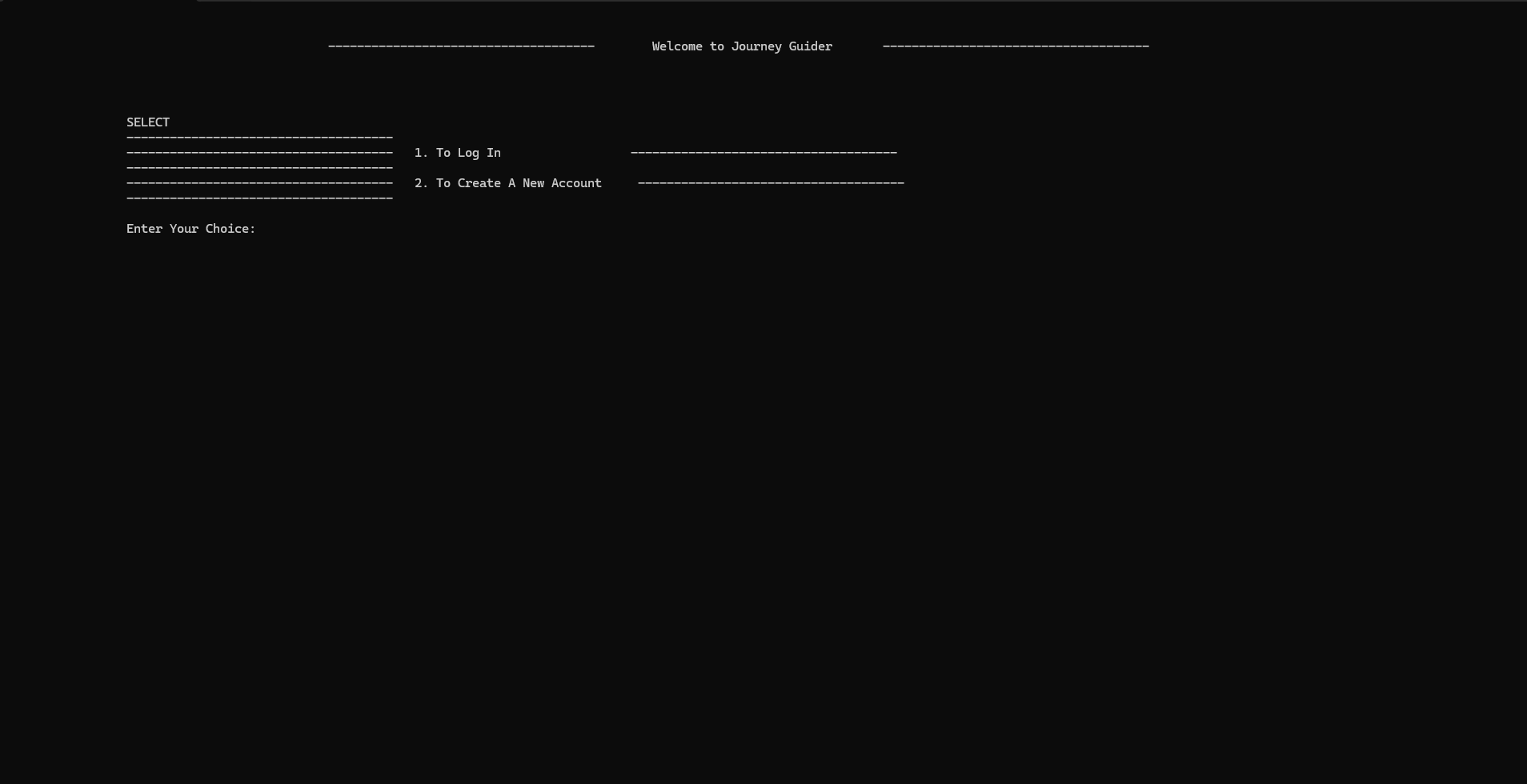


Figure 1-Main Screen

## **Account Creation:**

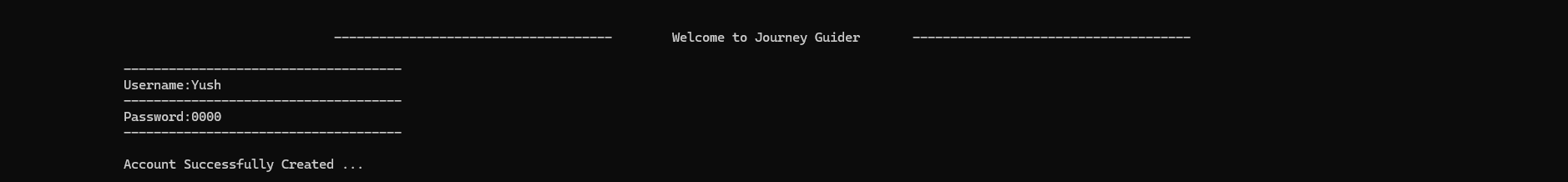


Figure 2-Sign Up Page

## **After Login Screen:**

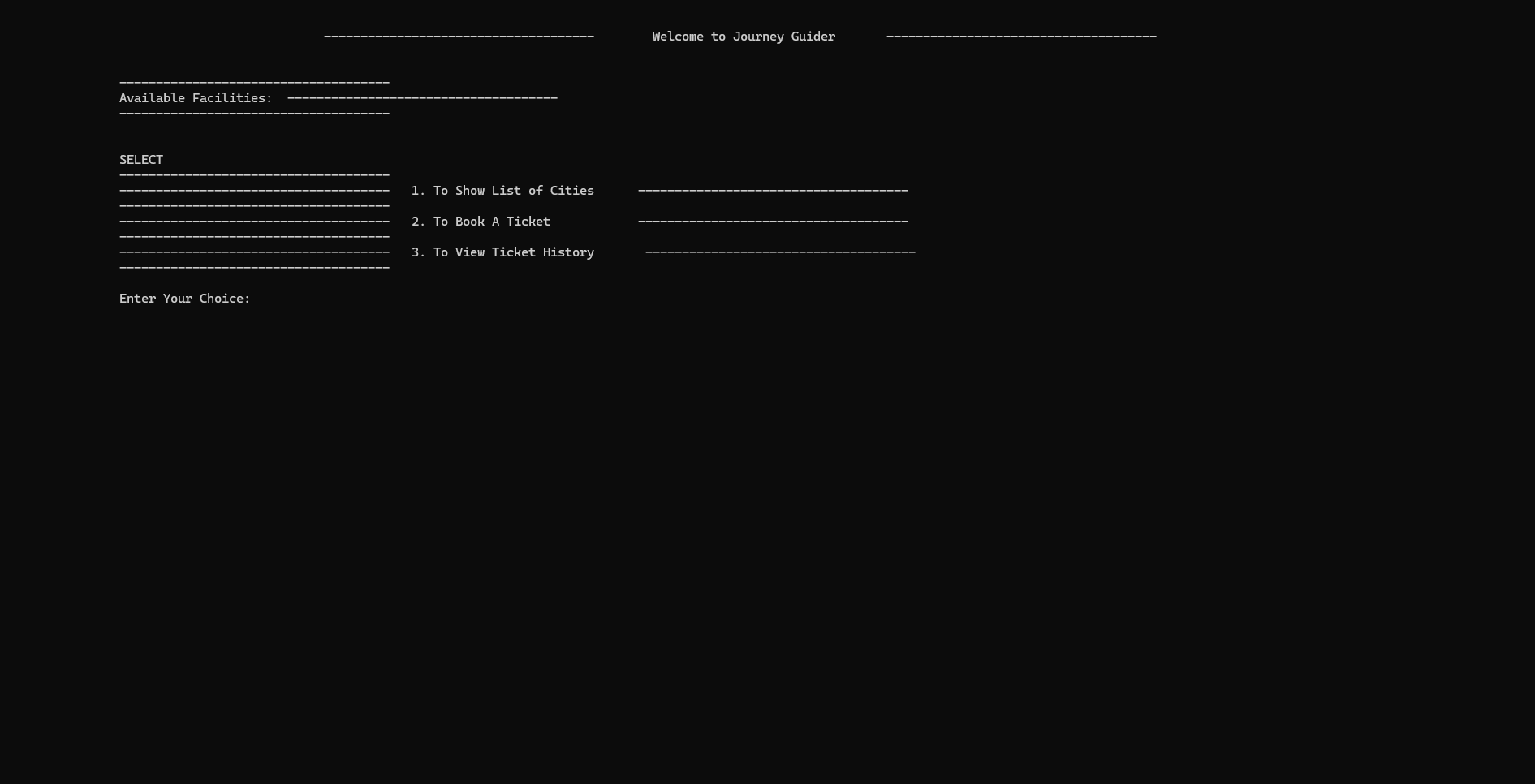


Figure 3-Available Facilities

## **Predefined Cities List:**

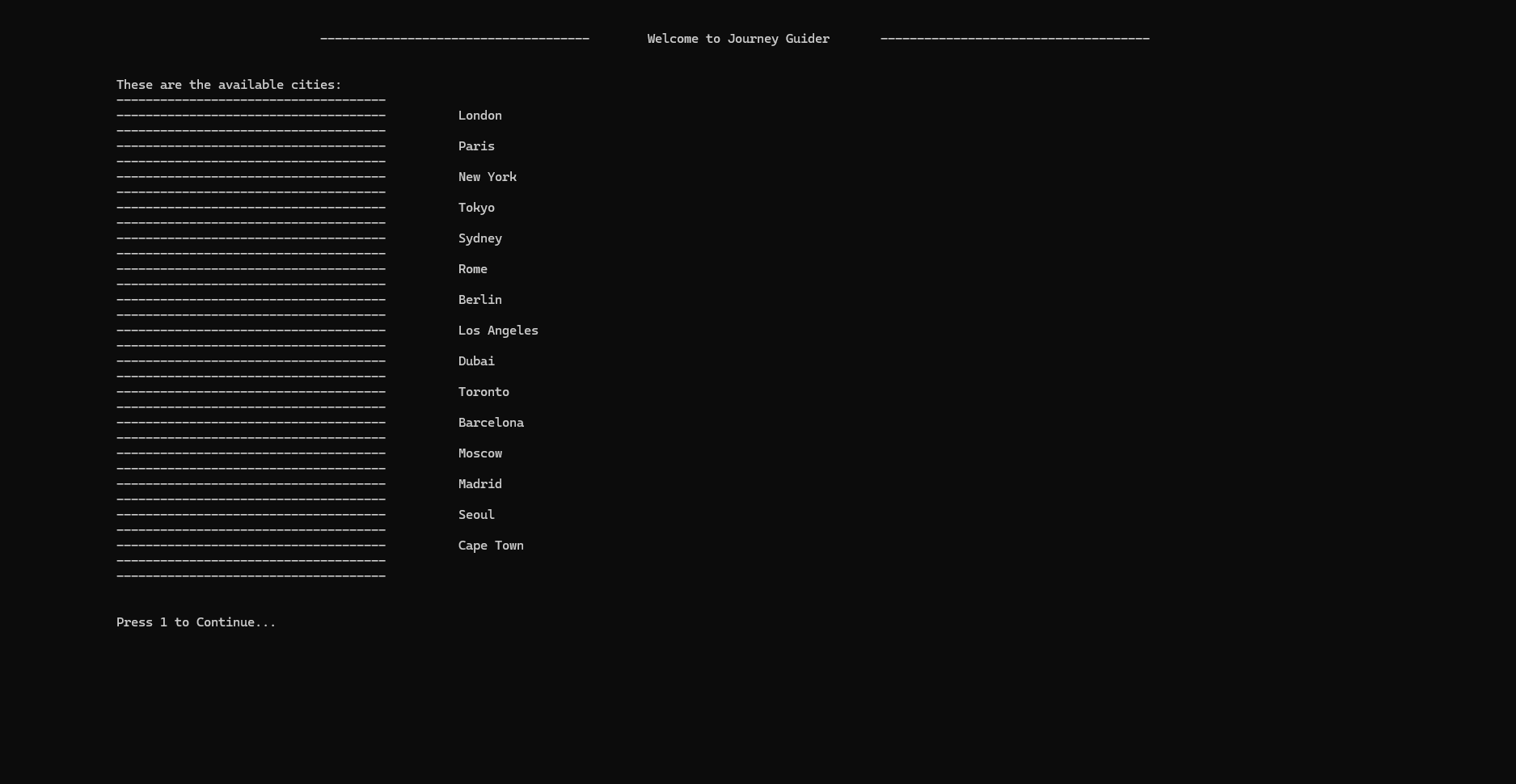


Figure 4-List Of Cities

## **Ticket Booking:**

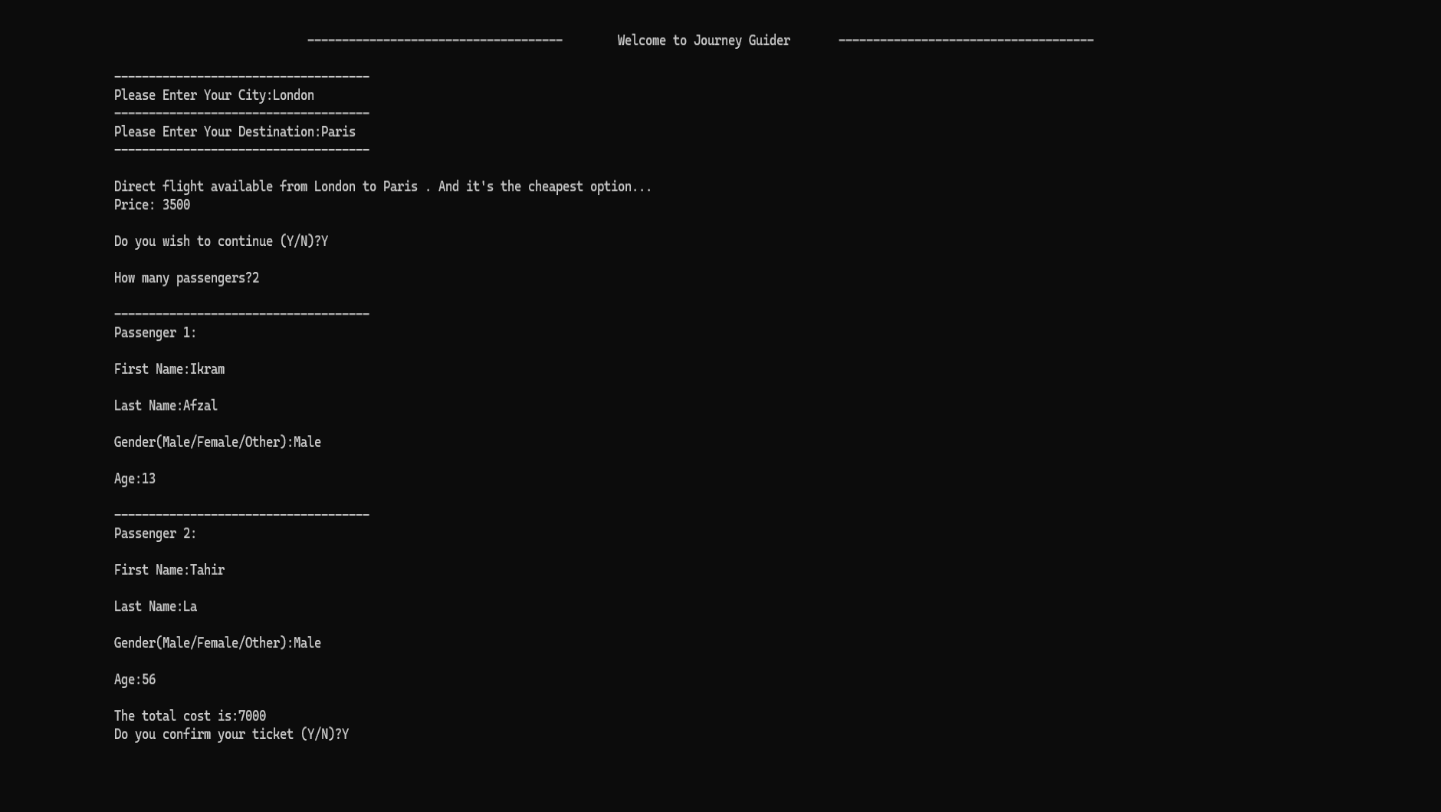


Figure 5-Ticket Booking

## **Display Price:**

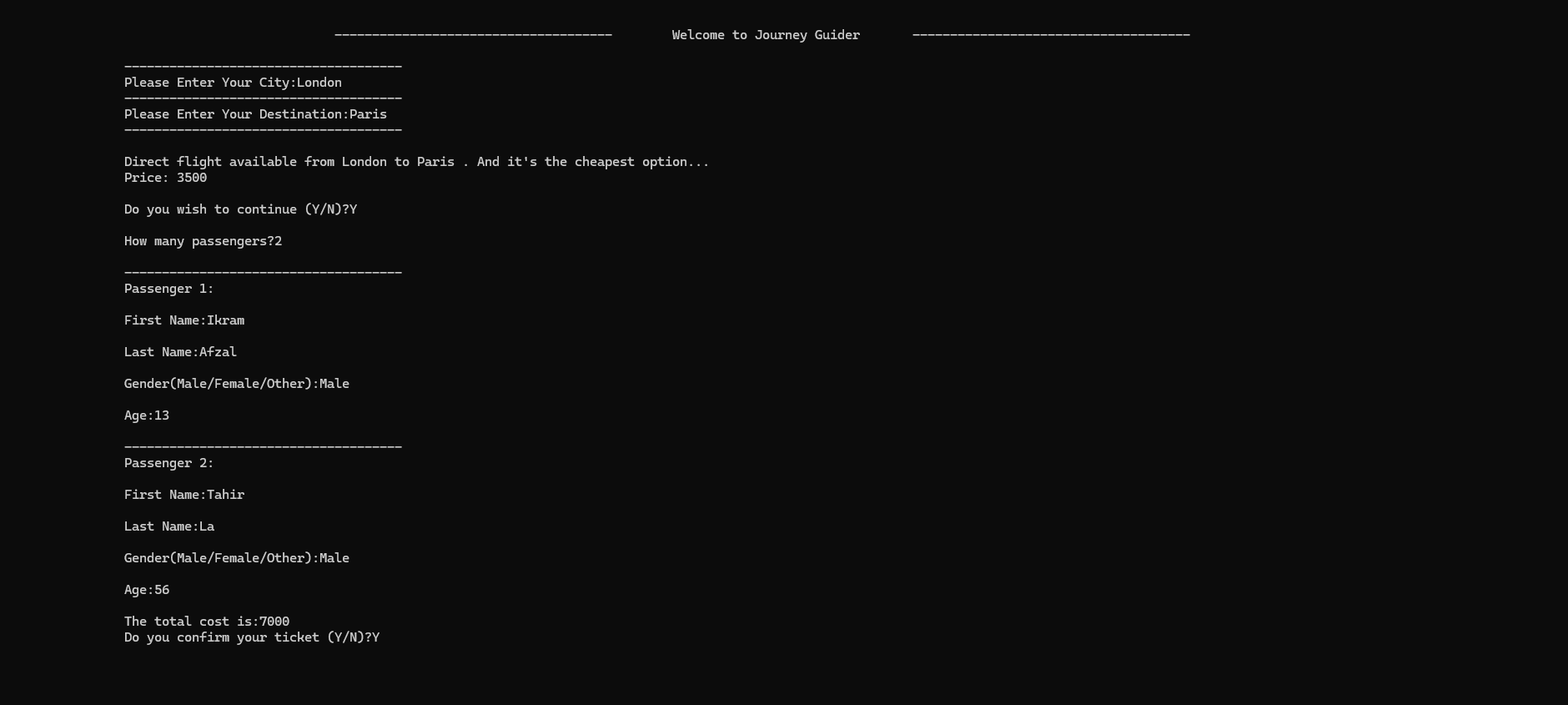


Figure 6-Price

## **E-Ticket Generation:**

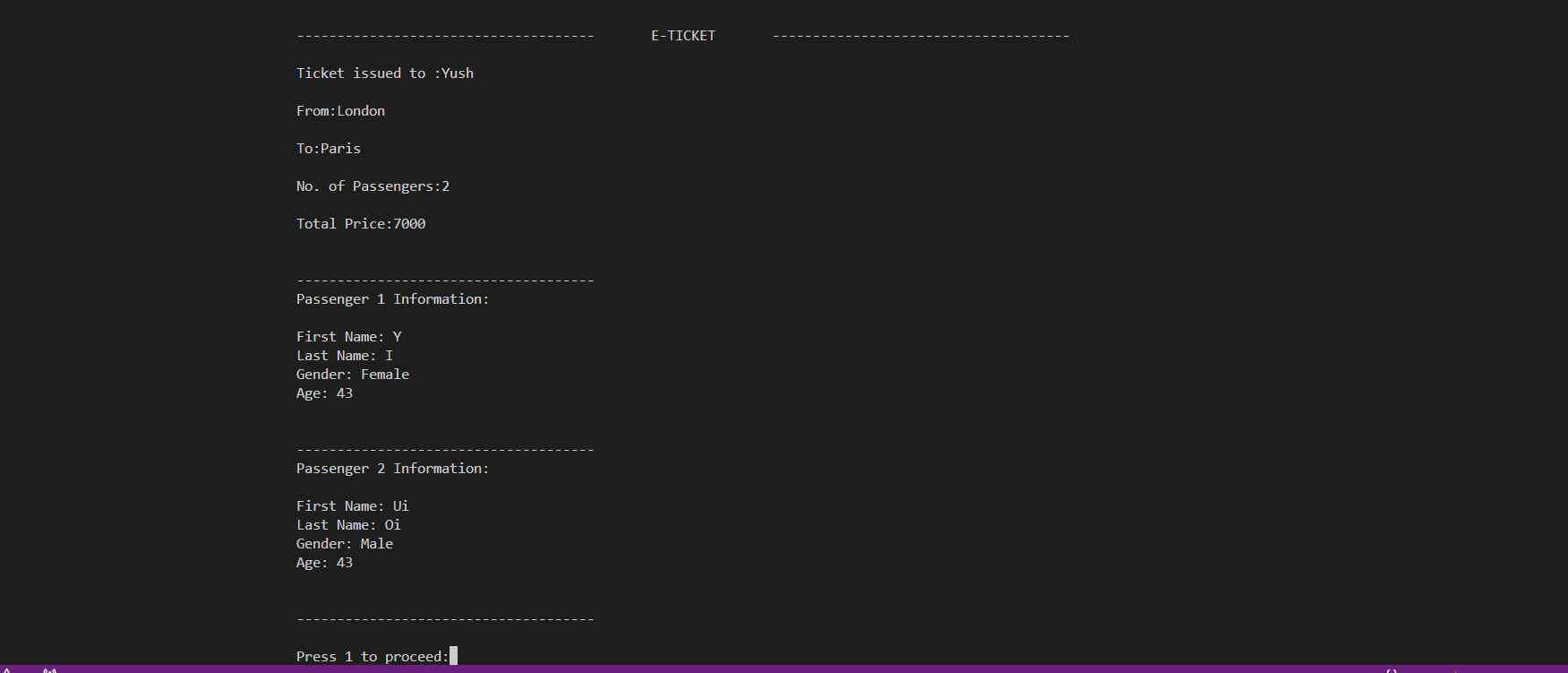


Figure 7-E-Ticket

## **Ticket History of Logged-In User:**



Figure 8-Ticket Generation

# **Languages and Header Files Used:**

## **Languages:**

* **C++**: The project is implemented in C++, using its standard libraries for efficient data management and algorithms.

## **Header Files:**

1. **<iostream>**: Provides input/output functionality, allowing user interaction with the console (e.g., cin, cout).
2. **<cstring>**: Offers functions for C-style string manipulation, such as strcmp() and strcpy().
3. **<fstream>**: Enables file handling, allowing reading from and writing to files (e.g., ifstream, ofstream).
4. **<vector>**: Defines the vector container, providing dynamic arrays to store collections of data like cities or distances.
5. **<algorithm>**: Provides algorithms for sorting, searching, and modifying data in containers (e.g., sort(), find()).
6. **<string>**: Introduces the string class for easier and safer string manipulation compared to C-style strings.
7. **#define INF 999999999**: A macro representing "infinity" for unreachable paths in algorithms like Bellman-Ford and Warshall.

These header files facilitate input/output, string handling, file operations, dynamic data management, algorithmic processing, and efficient pathfinding in the project.

# **Future Enhancements:**

Looking ahead, we plan to significantly improve the project by introducing a graphical user interface (GUI). This enhancement will provide a more engaging and user-friendly experience, enabling users to interact with the application in a more intuitive and visually appealing manner. The GUI will include interactive elements such as clickable buttons, dropdown menus, and dynamic maps, making the journey planning process smoother and more enjoyable.

In addition to the GUI, we also aim to integrate a weather feature into the application. This feature will allow users to receive real-time weather updates for their selected cities and routes, helping them make better-informed travel decisions. By providing weather forecasts, the project will enhance its functionality, offering travelers a more comprehensive tool for planning their trips, ensuring they can anticipate and adapt to weather conditions during their journeys. These enhancements will not only improve the user experience but also expand the project’s capabilities, making it a more valuable resource for travel planning.