Pakistan Railways Management System

Session 2023 - 2027

Submitted by:

Abdul Rehman 2023-CS-73

Supervised by:

Dr. Muhammad Awais Hassan

Course:

CSC-102 Programming Fundamentals

Department of Computer Science

University of Engineering and Technology, Lahore Pakistan

Table of contents

Contents

1. Project description

- The main purpose of this project is to develop a railways management system which can be used to manage the train routes and ticketing system. This system will allow the administrators to add or remove data of trains and they can easily calculate and gather data related to the trains and booked tickets.
- In the field of Computer Science, this application will showcase a handy use of file handling for building console-based applications in C++ for solving real-world problems and their professional use.

2. Users of Application

There are three types of users in the application based upon their role. The users include:

- **Admin:** Admin can specifically manage employee's data and can see the stats of the trains (No. of booked tickets, their revenue, total revenue). He has the access to all the data and functionalities in the application.
- **Employees:** Employees can access passenger's data and can make changes to it. They can add or remove passenger and can also add or delete train data.
- **Passenger:** Passenger can book or cancel a ticket. He can view the timetable of the trains and can also view the details or the ticket booked.

3. Functional Requirements

Functional requirements for each user are:

3.1. Admin functional requirements

Functional requirements of the user as an admin are:

Admin	Manage Employees Data	Add an employee's data in the application so that he can login to application and can work for admin.
		Delete an employee's data if he wants to remove any employee, can delete it.
		Update an employee's data. In case he wants to change any credential of the employee can do it easily. But userID cannot be change.
		View all employee's data so that employee's data is displayed in tabular form

		Search an employee's data. If data exists, he can view it.
	Manage Passengers Data	Add a passenger's data in the application so that he can login and access the services of railways.
		Delete a passenger's data if he wants to remove any passenger, can delete it.
		Update a passenger's data. In case he wants to change any credential of the passenger can do it easily. But userID cannot be change.
		View all passenger's data so that passenger's data is displayed in a tabular form.
		Search a passenger's data. If data exists, he can view it.
	Manage Train Routes	Add a train route so that user can book ticket and can avail the services of railways.
		Delete a train route. In case there occurs, some unexpected incident admin has the authority to delete a train's route
		Can view all the train routes available in a tabular form so that can easily manage further trains.
	Manage Ticketing System	Book a ticket for a passenger if ticket is not already booked.
		Cancel an already booked ticket of a passenger.
		View all the booked tickets in a table form.
	Tickets Details	Views total number of tickets booked of each train and also revenue collected from each train and total revenue and booked tickets.

3.2.Employees functional requirements

Functional requirements of the user as an employee are:

Employee	Manage Passengers Data	Add a passenger's data in the application so that he can login and access the services of railways.
		Delete a passenger's data if he wants to remove any passenger, can delete it.
		Update a passenger's data. In case he wants to change any credential of the passenger can do it easily. But userID cannot be change.
		View all passenger's data so that passenger's data is displayed in a tabular form.
		Search a passenger's data. If data exists, he can view it.
	Manage Train	Add a train route so that passenger can book ticket and can avail the services of railways.
		Delete a train route. In case there occurs, some unexpected incident admin has the authority to delete a train's route

	Routes	Can view all the train routes available in a tabular form so that can easily manage further trains.
	Manage Ticketing System	Book a ticket for a passenger if ticket is not already booked.
		Cancel an already booked ticket of a passenger.
		View all the booked tickets in a table form.
	Change Password	An employee can change its password by entering current password then is allowed to enter new password.
	View Data	An employee can view its own data.

3.3. Passengers' functional requirements

Functional requirements of the user as a passenger are:

Passenger	Book Ticket	Can book a ticket in train available. If ticket already booked, he cannot book a ticket again!
	Cancel Ticket	A passenger can change its password by entering current password then is allowed to enter new password.
	View Trains Available	A passenger can view the data of all the trains available.
	View Data	A passenger can view its own data.
	View Ticket Details	A passenger can view the details of the ticket booked.
	Change Password	A passenger can change its password by entering current password then is allowed to enter new password.

4. Wireframes

Wireframes of the railways management system are:

Figure 1: Start Menu

Figure 2: User SignUp

Figure 3: User SignIn

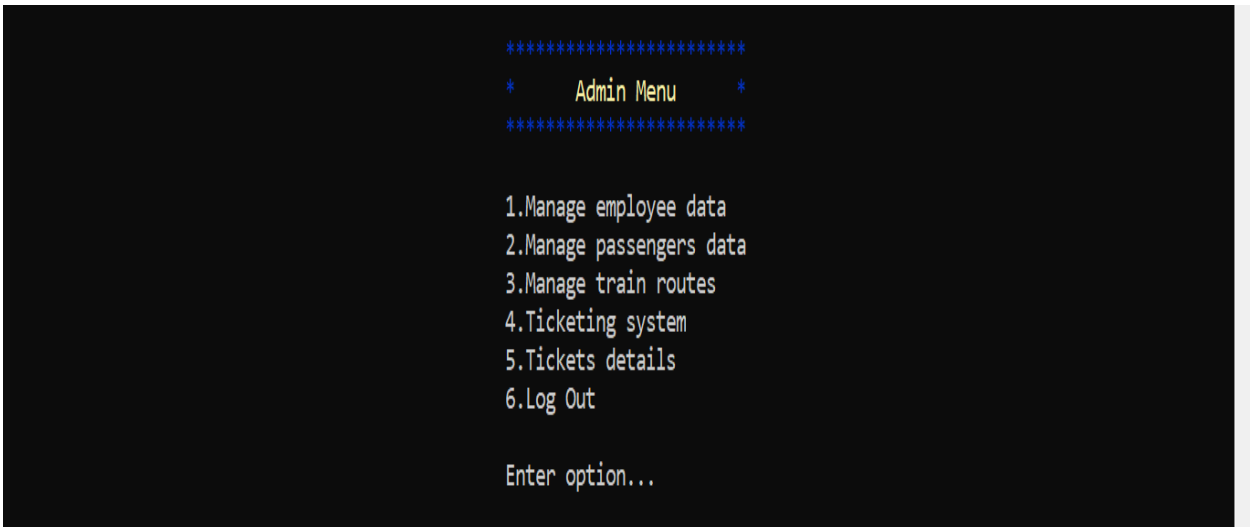


Figure 4: Admin Menu

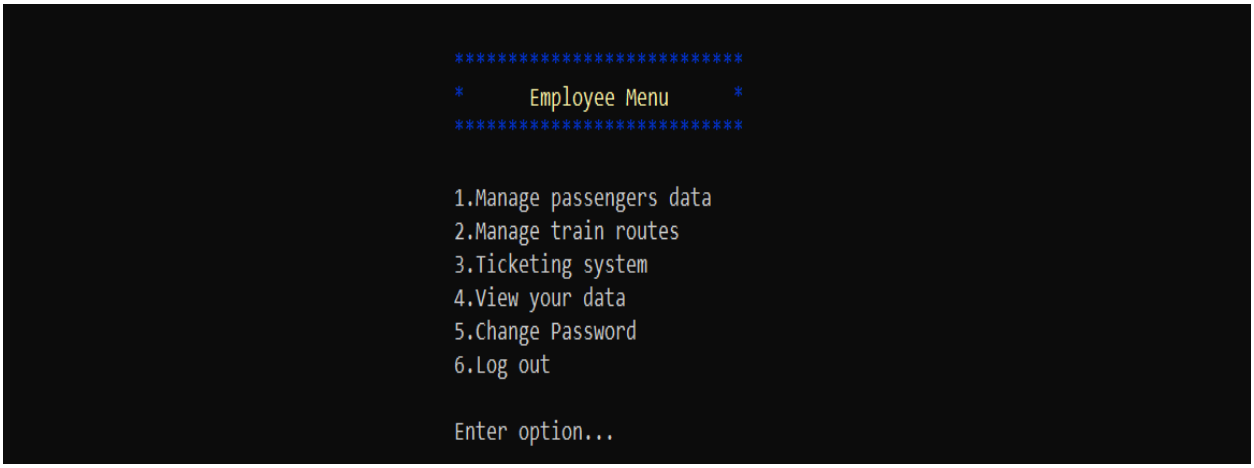


Figure 5: Employee Menu

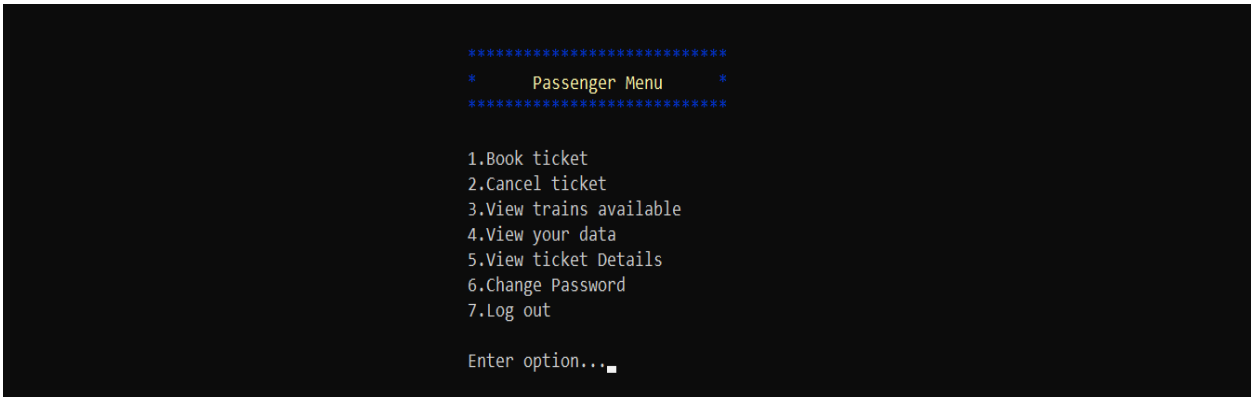


Figure 6: Passenger Menu

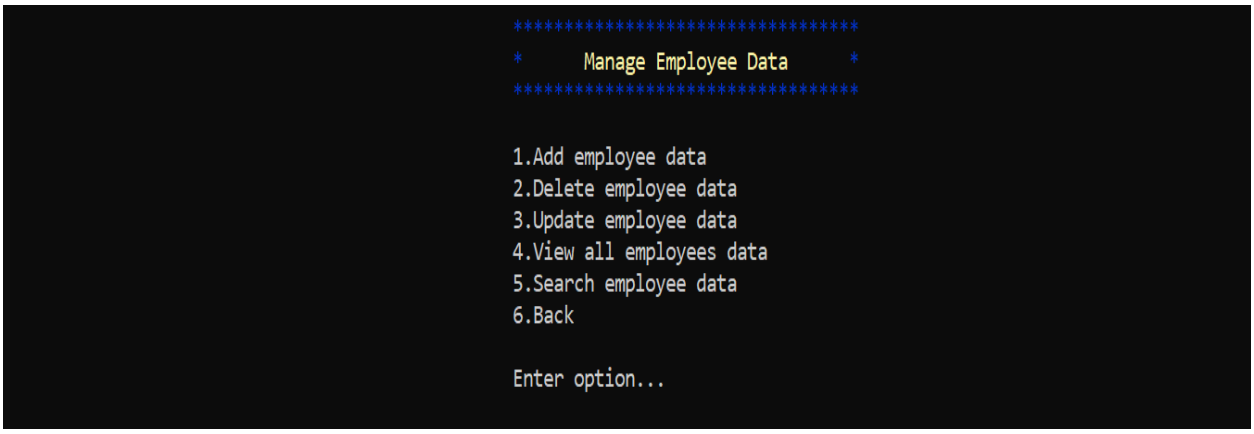


Figure 7: Manage Employee Menu

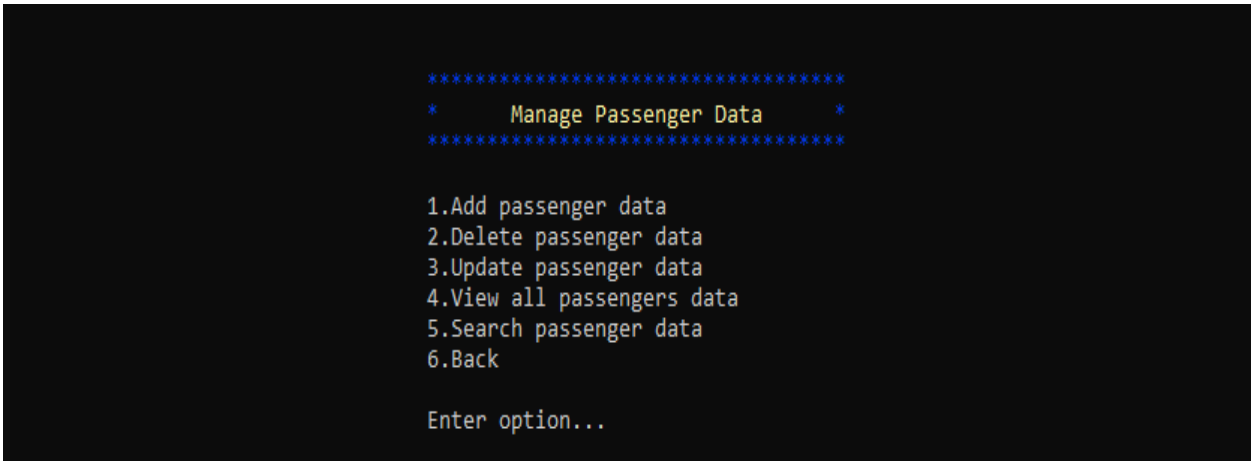


Figure 8: Manage Passenger Menu

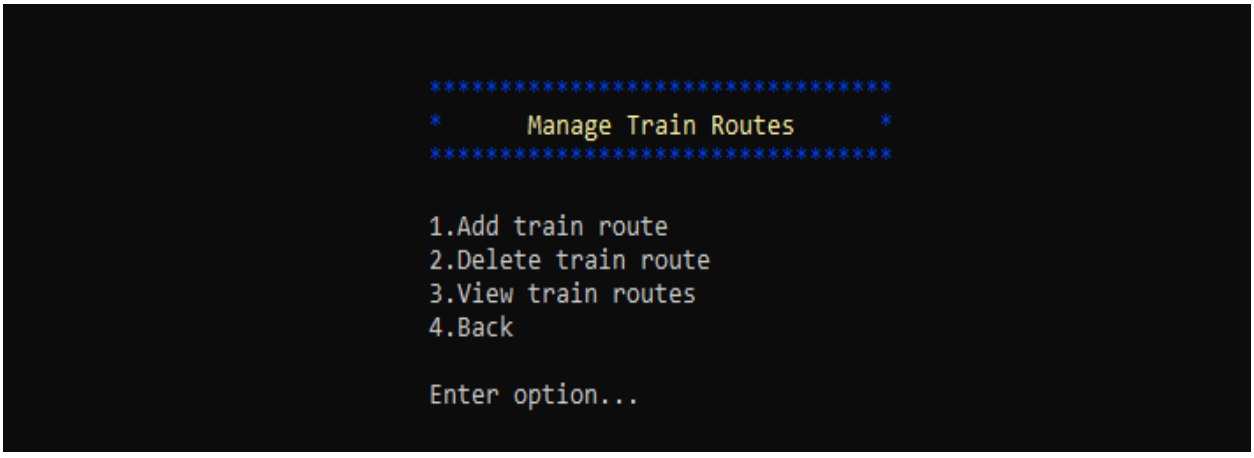


Figure 9: Manage Train Routes Menu

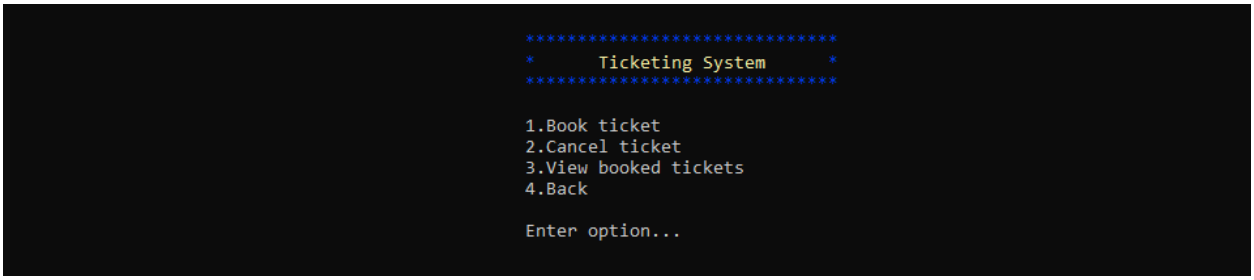


Figure 10: Manage Ticketing System Menu

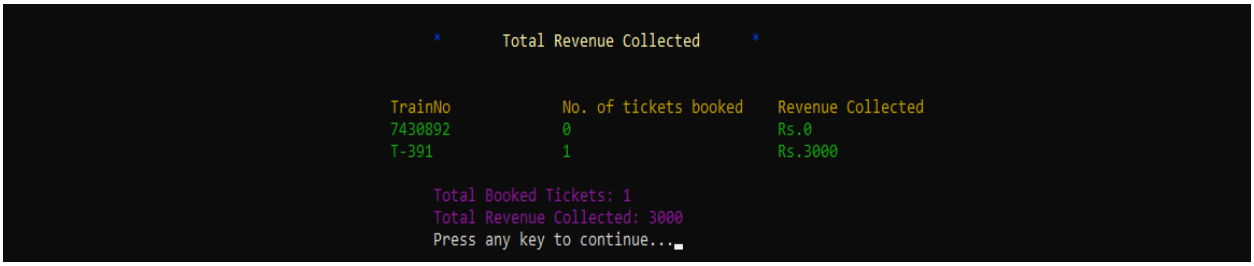


Figure 11: Total Revenue

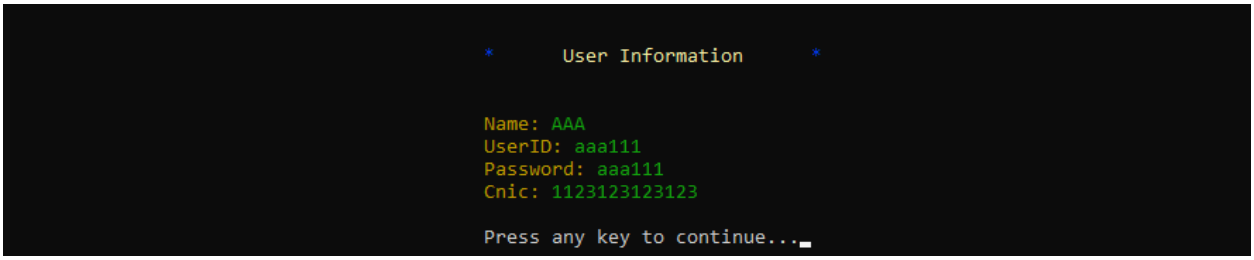


Figure 12: View User Information

5. Data Structures (Parallel Arrays)

The parallel arrays along with counter variables are:

5.1.Passengers Data

```
string passengerName[1000];
string passengerID[1000];
string passengerIDPassword[1000];
string passengerCnic[1000];
int passengerCountIdx = 0;
```

```
string passengerNameSU;
bool passengerNameCheck;
string userIDSU;
bool userIDCheckSU;
string userPasswordSU;
bool validation;
string userCnicSU;
bool cnicCheck;
int pasgIdx;
string pasgPasswordSI;
```

5.2.Passenger Ticket Data

```
string passengerTicketStatus[1000];
string passengerTrainNo[1000];
string passengerTicketRoute[1000];
string passengerArrivalCity[1000];
string passengerDepartureCity[1000];
int passengerTicketPrice[1000];
string routeNo;
int indexTrain;
```

5.3.Employee Tick Data

```
string employeeName[100];
string employeeID[100];
string employeeIDPassword[100];
string employeeCnic[100];
int employeeCountIdx = 0;
string empNamein;
bool empNameCheck;
string empIdin;
bool empIDCheck;
string empPasswordin;
bool passValidation;
string empCnicin;
bool cnicCheck;
int empIdx;
string empPasswordSI;
```

5.4.Train Data

```
string trainNo[100];
string trainArrivalCity[100];
string trainDepartureCity[100];
string trainRoute[100];
int trainTicketPrice[100];
int trainCountIdx = 0;
string trainId;
string departureCity;
string arrivalCity;
string price;
string trainCode;
int totalRevenue = 0;
int totalTicketsSold = 0;
int numOfBookedTickets[trainCountIdx];
int revenueOfEachTrain[trainCountIdx];
```

5.5. Variables used for all users

```
string ID;
string newPassword;
string userIDSI;
string role;
```

6. Function Prototypes

All the functions used in the application are:

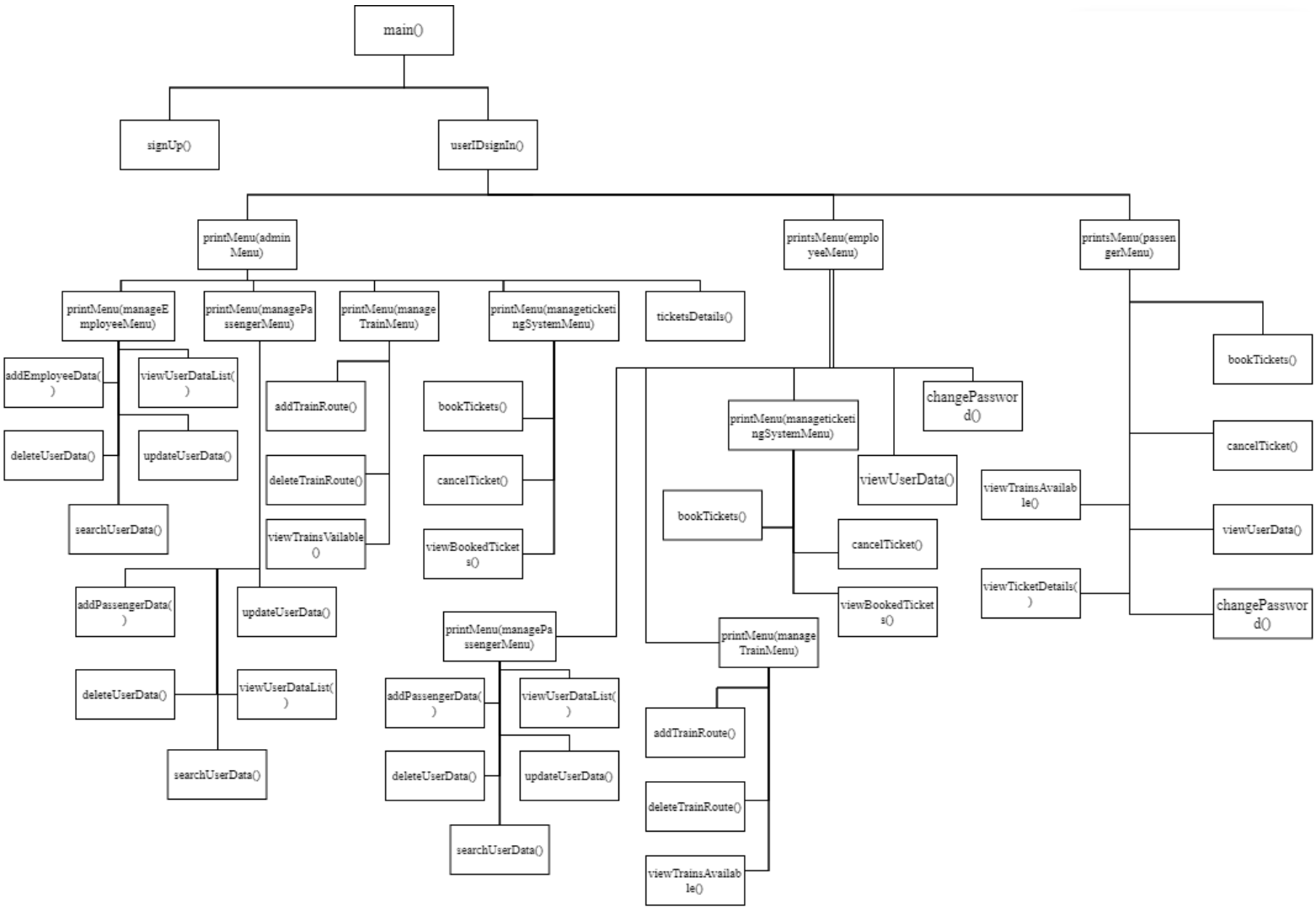
```
void header();
void startHeader();
void startMenuHeader();
void signUpHeader();
void signInHeader();
void adminHeader();
void employeeHeader();
void passengerHeader();
void manageEmployeeHeader();
void managePassengerHeader();
void manageTrainsHeader();
void manageTicketetingHeader();
void printSubHeader(string);
void noteUserName();
void noteuserIDpassenger();
```

```
void noteSUpassword();
void noteSUcnic();
void noteAddEmployee();
void noteAddTrain();
void noteDepartureCity(string[]);
void noteArrivalCity(string, string[]);
void noteRoutesavail(string[], string[], int);
void eraseInstruction();
string printMenu(string[], int);
string userNameSignUp(int);
bool userNameValidationCheck(string);
string userIDSignup(string[], int);
bool userIDCheckSignup(string, string[], int);
string userPasswordSignup(int);
bool passwordValidationCheckSignup(string);
string userCnicSignup(string[], string[], int, int, int);
bool userCnicValidationSignup(string, string[], string[], int, int);
void saveSUInformation(string, string[], string, string[], string, string[], string, string[],
int &);
string userIDSignIn(string[], string[], int, int);
bool userCheckSignIn(string, string[], string[], int, int);
string adminPasswordCheck();
bool userCheck(string, string[], int);
string roleCheck(string);
int indexCheck(string, string[], int);
string userPasswordSignIn(string);
void addEmployeeData(string[], string[], string[], string[], string[], int &, int);
string empUserIDInput(string[], int);
bool employeeIDCheck(string, string[], int);
void addPassengerData(string[], string[], string[], string[], string[], string[], int &, int);
void deleteUserData(string[], string[], string[], string[], string, int &);
void deleteData(string[], string[], string[], string[], int &, int);
string YesNoChoice(string);
void deleteTicketData(string[], string[], string[], string[], string[], int[], int, int);
void updateUserData(string[], string[], string[], string[], string[], string, int, int);
string updateData(string[], string[], string[], string[], int, int, int, string);
string updateDataChoice();
string changePassword(string[], int);
void viewUserDataList(string[], string[], string[], string[], string, int);
void viewUserData(string[], string[], string[], string[], int, int, int);
void searchUserData(string[], string[], string[], string[], string, int);
void ticketStatusPassenger(string[], string[], string[], string[], string[], int[], int);
void addTrainRoute(string[], string[], string[], string[], int[], int &, string[]);
```

```
string trainNoInput(string[], int);
bool trainNoValidation(string, string[], int);
bool trainCheck(string, string[], int);
string trainArrivalCityInput(string, string[]);
string trainDepartureCityInput(string[]);
bool cityNameValidation(string, string[]);
int trainTicketPriceIn();
bool ticketPriceValidation(string);
int stringToIntConversion(string);
string deleteTrainRoute(string[], string[], string[], string[], int[], int &);
void deleteTrainTicketDetails(string[], string[], string[], string[], string[], int[], int,string);
void deleteData(string[], string[], string[], string[], int[], int, int &);
void viewTrainsAvailable(string[], string[], string[], string[], int[], int);
void bookTickets(string[], string[], string[], string[], string[], string[], string[], string[],
string[], int[], int[], int, int);
void saveTicketData(string[], string[], string[], string[], string[], string[], string[], string[],
string[], int[], int[], int, int);
void cancelTicket(string[], string[], string[], string[], string[], int[], int);
void viewTicketDetails(string[], string[], string[], string[], int[], int);
void viewBookedTickets(string[],string[],string[],string[],string[],string[],string[],int[],
int);
void ticketsDetails(string[], string[], int[], int[], int, int);
void employeesNewDataFile(string[], string[], string[], string[], int);
void employeesDataUpdateFile(string[], string[], string[], string[], int);
void employeeDataLoad(string[], string[], string[], string[], int &);
void passengersNewDataFile(string[], string[], string[], string[], string[], string[], string[],
string[], string[], int[], int);
void passengersDataUpdateFile(string[], string[], string[], string[], string[], string[],
string[], string[], string[], int[], int);
void passengersDataLoad(string[], string[], string[], string[], string[], string[], string[],
string[], string[], int[], int &);
void trainsNewDataFile(string[], string[], string[], string[], int[], int);
void trainsDataUpdateFile(string[], string[], string[], string[], int[], int);
void trainsDataLoad(string[], string[], string[], string[], int[], int &);
string loadUserAttribute(string, int &);
string userIDInput(int i, int y);
string inputs();
void pressAnyKey(int x, int y);
void space(int x, int y);
void headerCls();
void printStatement(string, int, int, string);
```

7. Functions Working Flow

The flow of functions among the app is controlled as following:



8. Complete Code of the Business Application

```
#include <iostream>
#include <conio.h>
#include <windows.h>
#include <iomanip>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <fstream>

using namespace std;

// headers
void header();
void startHeader();
void startMenuHeader();
void signUpHeader();
void signInHeader();
void adminHeader();
void employeeHeader();
void passengerHeader();
void manageEmployeeHeader();
void managePassengerHeader();
void manageTrainsHeader();
void manageTicketetingHeader();
void printSubHeader(string);
```

```
// guidance notes
void noteUserName();                    // user Name instructions
void noteuserIDpassenger();             // user ID instructions
void noteSUpassword();                  // password instructions
void noteSUcnic();                      // user cnic instructions
void noteAddEmployee();                 // employee id instructions
void noteAddTrain();                    // trainNo instructions
void noteDepartureCity(string[]);       // departureCity instructions
void noteArrivalCity(string, string[]); // arrival city instrucitons
void noteRoutesavail(string[], string[], int); // displays the routes available
void eraseInstruction();                // erases notes

// menus
string printMenu(string[], int); // print the menus

// user signUp
string userNameSignUp(int);                                  // user Name input
bool userNameValidationCheck(string);                        // checks name validation
string userIDSignup(string[], int);                          // user ID input
bool userIDCheckSignup(string, string[], int);               // checks userID validation
string userPasswordSignup(int);                              // ID password input
bool passwordValidationCheckSignup(string);                  // checks password validation
string userCnicSignup(string[], string[], int, int, int);    // cnic input
```

bool userCnicValidationSignup(string, string[], string[], int, int); // checks cnic validation

void saveSUInformation(string, string[], string, string[], string, string[], string, string[], int &); // saves user information

// user signIn

string userIDSignIn(string[], string[], int, int); // user ID input

bool userCheckSignIn(string, string[], string[], int, int); // checks whether a user is present or not

string adminPasswordCheck(); // takes input for Admin Password

bool userCheck(string, string[], int); // checks if userID is present in the provded passengers or employee data

string roleCheck(string); // Checks role of User ID

int indexCheck(string, string[], int); // index of User ID

string userPasswordSignIn(string); // user ID password input

// adding employees

void addEmployeeData(string[], string[], string[], string[], string[], int &, int); // add employee data

string empUserIDInput(string[], int); // employee ID input

bool employeeIDCheck(string, string[], int); // checks employee ID validation

// adding passengers

void addPassengerData(string[], string[], string[], string[], string[], string[], int &, int);

// users data operations

void deleteUserData(string[], string[], string[], string[], string, int &); // takes userID Input calls other function for ID index check

void deleteData(string[], string[], string[], string[], int &, int); // deletes user data if ID is found and admin authenticates it

string YesNoChoice(string); // asks for permission for deleting data

void deleteTicketData(string[], string[], string[], string[], string[], int[], int, int); // deletes passengers ticket details along with data deletion

void updateUserData(string[], string[], string[], string[], string[], string, int, int); // call other functions, passes inputs to other functions for updating data

string updateData(string[], string[], string[], string[], int, int, int, string); // call other functions,takes input and pass it for saving data

string updateDataChoice(); // asks the user choice for changing data

string changePassword(string[], int); // change password

void viewUserDataList(string[], string[], string[], string[], string, int); // view all users data

void viewUserData(string[], string[], string[], string[], int, int, int); // view user data

void searchUserData(string[], string[], string[], string[], string, int); // searches for data of a userID

void ticketStatusPassenger(string[], string[], string[], string[], string[], int[], int); // saves initial ticket status or ticket status after cancelling ticket

// trains control

void addTrainRoute(string[], string[], string[], string[], int[], int &, string[]); // add train route

string trainNoInput(string[], int); // takes trainNo as input

bool trainNoValidation(string, string[], int); // checks if trainNo already exists or is valid

bool trainCheck(string, string[], int); // checks if train exists or not

string trainArrivalCityInput(string, string[]); // takes inputs for arrival city

string trainDepartureCityInput(string[]); // takes inputs for departure city

```
bool cityNameValidation(string, string[]);                                  // checks
the city name validations

int trainTicketPriceIn();                                                   // takes ticket
price as input

bool ticketPriceValidation(string);                                         // checks if
the price entered in string is able to be converted into float or int

int stringToIntConversion(string);                                          // converts
the price into float

string deleteTrainRoute(string[], string[], string[], string[], int[], int &);          //
takes inputs and asks for permission and pass data to funciton to delete

void deleteTrainTicketDetails(string[], string[], string[], string[], string[], int[], int,
string); // delete passenger tickets booked on a train if train data deleted

void deleteData(string[], string[], string[], string[], int[], int, int &);             //
deltes train data on confirmation

void viewTrainsAvailable(string[], string[], string[], string[], int[], int);           //
displays the trains available


// ticketing system

void bookTickets(string[], string[], string[], string[], string[], string[], string[], string[],
string[], int[], int[], int, int);    // book tickets for a passenger

void saveTicketData(string[], string[], string[], string[], string[], string[], string[], string[],
string[], int[], int[], int, int); // saves the data of the added train

void cancelTicket(string[], string[], string[], string[], string[], int[], int);
// cancel a booked ticket for a passenger

void viewTicketDetails(string[], string[], string[], string[], int[], int);
// displays the details of ticket of a passenger

void viewBookedTickets(string[], string[], string[], string[], string[], string[], string[],
int[], int);                    // displays all booked tickets data

void ticketsDetails(string[], string[], int[], int[], int, int);
// calculate and displays the total revenue collected


// file handeling employees
```

```
void employeesNewDataFile(string[], string[], string[], string[], int);    // saves data of new added employee to file

void employeesDataUpdateFile(string[], string[], string[], string[], int); // updates the file of employee data if an employee is deleted or its data is updated

void employeeDataLoad(string[], string[], string[], string[], int &);      // loads the employee data into the arrays


// file handeling passengers

void passengersNewDataFile(string[], string[], string[], string[], string[], string[], string[], string[], string[], int[], int);    // saves data of new added passenger to file

void passengersDataUpdateFile(string[], string[], string[], string[], string[], string[], string[], string[], string[], int[], int); // updates the file of passenger data if a passenger is deleted or its data is updated

void passengersDataLoad(string[], string[], string[], string[], string[], string[], string[], string[], string[], int[], int &);     // loads the passenger data into the arrays


// file handeling trains

void trainsNewDataFile(string[], string[], string[], string[], int[], int);    // saves data of new added train route to file

void trainsDataUpdateFile(string[], string[], string[], string[], int[], int); // updates the file of train data if a train route is deleted

void trainsDataLoad(string[], string[], string[], string[], int[], int &);     // loads the train data into the arrays


string loadUserAttribute(string, int &); // loads single attribute data


// redundant functions

string userIDInput(int i, int y); // for taking username inputs

string inputs();                  // for taking input

void pressAnyKey(int x, int y);   // gets key to continue

void space(int x, int y);         // erases a statement if input at a point is to be takes again

void headerCls();                 // clears screen and prints header
```

```
void printStatement(string, int, int, string); // prints statements

void gotoxy(int i, int j);

main()
{
   // Passengers data
   string passengerName[1000];       // name of each user
   string passengerID[1000];         // usernames for each passenger
   string passengerIDPassword[1000]; // passwords for each passenger
   string passengerCnic[1000];       // cnic of each passenger
   int passengerCountIdx = 0;        // number of passengers who have signed up

   // Passengers Tickets Details
   string passengerTicketStatus[1000];  // ticket status of the passenger
   string passengerTrainNo[1000];       // train no on which passenger has the ticket
   string passengerTicketRoute[1000];   // route for the ticket booked
   string passengerArrivalCity[1000];   // the city to which train is going
   string passengerDepartureCity[1000]; // the city from which train is leaving
   int passengerTicketPrice[1000];      // price of the ticket booked by the passenger

   // Employees data
   string employeeName[100];       // name of each employee
   string employeeID[100];         // usernames for each emmployee
   string employeeIDPassword[100]; // passwords for each employee
   string employeeCnic[100];       // cnic for each employee
   int employeeCountIdx = 0;       // number of employees who have been added
```

```
    // Train details
    string cities[15] = {"Lahore", "Peshawar", "Quetta", "Rawalpindi", "Faisalabad", "Hyderabad", "Sialkot", "Karachi", "Islamabad", "Multan", "Gujranwala", "Okara", "Sahiwal", "Jehlum", "D.G.K"};
    string trainNo[100];            // code of the train
    string trainArrivalCity[100];   // arrival city of the train
    string trainDepartureCity[100]; // departure city of the train
    string trainRoute[100];         // route of the train
    int trainTicketPrice[100];      // ticket price of the train
    int trainSeats[100];            // no. of seats in each train
    int trainCountIdx = 0;          // no. of trains

    // Menus
    string startMenu[3] = {"Sign up", "Sign in", "Exit"};
    string adminMenu[6] = {"Manage employee data", "Manage passengers data", "Manage train routes", "Ticketing system", "Tickets details", "Log Out"};
    string manageEmployeeMenu[6] = {"Add employee data", "Delete employee data", "Update employee data", "View all employees data", "Search employee data", "Back"};
    string managePassengerMenu[6] = {"Add passenger data", "Delete passenger data", "Update passenger data", "View all passengers data", "Search passenger data", "Back"};
    string manageTrainMenu[4] = {"Add train route", "Delete train route", "View train routes", "Back"};
    string manageTicketSystemMenu[4] = {"Book ticket", "Cancel ticket", "View booked tickets", "Back"};
    string employeeMenu[6] = {"Manage passengers data", "Manage train routes", "Ticketing system", "View your data", "Change Password", "Log out"};
    string passengerMenu[7] = {"Book ticket", "Cancel ticket", "View trains available", "View your data", "View ticket Details", "Change Password", "Log out"};

    // Headers
```

```
    string headerNames[20] = {"Add Employee", "Delete Employee", "Update Employee
Data", "Employees Data", "Search Employee Data",

                              "Add Passenger", "Delete Passenger", "Update Passenger Data",
"Passengers Data", "Search Passenger Data",

                              "Add Train Route", "Remove Train Route", "Train Routes Available",
"Book Ticket", "Cancel Ticket",

                              "Booked Ticket Details", "User Information", "View Ticket Details", "
Total Revenue Collected", "Change Password"};


    // load data files for passengers,employees and trains

    passengersDataLoad(passengerName, passengerID, passengerIDPassword,
passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute,
passengerArrivalCity, passengerDepartureCity, passengerTicketPrice,
passengerCountIdx);

    employeeDataLoad(employeeName, employeeID, employeeIDPassword,
employeeCnic, employeeCountIdx);

    trainsDataLoad(trainNo, trainRoute, trainArrivalCity, trainDepartureCity,
trainTicketPrice, trainCountIdx);


    system("cls"); // clears screen

    startHeader(); // animated header


    string option;


    // loop terminates if option entered by user is 3

    while (true)

    {

        // prints headers

        headerCls();

        startMenuHeader();
```

```
        option = printMenu(startMenu, 3); // user choice

        if (option == "1") // if user want sign up enters 1
        {
            headerCls();
            signUpHeader();

            if (passengerCountIdx == 1000) // if username space not available
            {
                printStatement("Sorry you cannot sign up!", 50, 24, "W");
            }
            else // if username space is available
            {
                string passengerNameSU = userNameSignUp(23);                    // name of
user
                bool passengerNameCheck = userNameValidationCheck(passengerNameSU);
// checks if Name enterd by the user is in correct format or not

                if (passengerNameCheck == true)
                {

                    string userIDSU = userIDSignup(passengerID, passengerCountIdx);
// username for signing up
                    bool userIDCheckSU = userIDCheckSignup(userIDSU, passengerID,
passengerCountIdx); // if userID already exists return true otherwise false

                    if (userIDCheckSU == true) // if username already exists or wrong input
                    {
                        printStatement("Invalid UserID. Sign up failed!", 50, 25, "W");
                    }
```

```
            else // if username does not exist
            {
                string userPasswordSU = userPasswordSignup(25);                 // password for signing up
                bool validation = passwordValidationCheckSignup(userPasswordSU); // if password is valid returns true

                if (validation == true) // if passenger uses correct password format
                {
                    string userCnicSU = userCnicSignup(passengerCnic, employeeCnic, passengerCountIdx, employeeCountIdx, 26);              // cnic for signing up
                    bool cnicCheck = userCnicValidationSignup(userCnicSU, passengerCnic, employeeCnic, passengerCountIdx, employeeCountIdx); // checks if cnic is valid

                    if (cnicCheck == true) // if cnic is correct and is not already found
                    {
                        // saving info for sign up
                        ticketStatusPassenger(passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                        saveSUInformation(passengerNameSU, passengerName, userIDSU, passengerID, userPasswordSU, passengerIDPassword, userCnicSU, passengerCnic, passengerCountIdx);

                        passengersNewDataFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                        printStatement("Signed up successfully!", 50, 27, " ");
                    }
                    else // if cnic format is not correct or is already present
```

```
                    {
                        printStatement("Invalid Cnic. Sign up failed!", 50, 27, "W");
                    }
                }
                else // if user uses wrong format for password
                {
                    printStatement("Invalid Password format. Sign up failed!", 50, 26,
"W");
                }
            }
        }
        else // if user enters wrong name format
        {
            printStatement("Invalid Name format. Sign up failed!", 50, 24, "W");
        }
    }
}
else if (option == "2") // if user wants to sign in enters 2
{
    headerCls();
    signInHeader();

    string userIDSI = userIDSignIn(passengerID, employeeID, passengerCountIdx,
employeeCountIdx); // userID for signing in

    if (userIDSI == "admin") // if userID is admin
    {
        string adminPassword = adminPasswordCheck(); // checks password for admin
```

```
            if (adminPassword == "admin") // admin sign in successful
            {
                // loop terminates if admin enters 6
                while (true)
                {
                    headerCls();
                    adminHeader();

                    string choice = printMenu(adminMenu, 6); // takes admin's choice as input for submenu

                    if (choice == "1") // if admin enters 1 employee management menu appears
                    {
                        // loop terminates if user enters 5
                        while (true)
                        {
                            headerCls();
                            manageEmployeeHeader();

                            choice = printMenu(manageEmployeeMenu, 6); // takes user choice

                            if (choice == "1") // if admin wants to add an employee enters 1
                            {
                                headerCls();
                                printSubHeader(headerNames[0]);

                                // add an employee data
```

```
                addEmployeeData(employeeName, employeeID,
employeeIDPassword, employeeCnic, passengerCnic, employeeCountIdx,
passengerCountIdx);
            }
            else if (choice == "2") // if admin wants to delete employee enters 2
            {
                headerCls();
                printSubHeader(headerNames[1]);

                int x = employeeCountIdx; // saves employee count temporarily

                // delete an employee data
                deleteUserData(employeeName, employeeID,
employeeIDPassword, employeeCnic, "Employee", employeeCountIdx);

                if (x != employeeCountIdx) // if employee data is deleted
                {
                    // employee file is updated
                    employeesDataUpdateFile(employeeName, employeeID,
employeeIDPassword, employeeCnic, employeeCountIdx);
                }
            }
            else if (choice == "3") // if admin wants to update employee enters 3
            {
                headerCls();
                printSubHeader(headerNames[2]);

                // update an employee data
```

```
                updateUserData(employeeName, employeeID,
employeeIDPassword, employeeCnic, passengerCnic, "Employee", employeeCountIdx,
passengerCountIdx);


                // updates employee data file after the employee data is updated
                employeesDataUpdateFile(employeeName, employeeID,
employeeIDPassword, employeeCnic, employeeCountIdx);
            }
            else if (choice == "4") // if admin wants to view employee enters 4
            {
                headerCls();
                printSubHeader(headerNames[3]);


                // displays all employees data
                viewUserDataList(employeeName, employeeID,
employeeIDPassword, employeeCnic, "Employee", employeeCountIdx);
            }
            else if (choice == "5") // if admin wants to search an employee enters
5
            {
                headerCls();
                printSubHeader(headerNames[4]);


                // searches employee data for employeeID entered by admin
                searchUserData(employeeName, employeeID,
employeeIDPassword, employeeCnic, "Employee", employeeCountIdx);
            }
            else if (choice == "6") // if admin wants to go back enters 6
            {
                break;
```

```
            }
            else // if admin enters wrong input
            {
                printStatement("Wrong userChoice!...", 50, 30, "W");
            }
        }
    }
    else if (choice == "2") // if admin enters 2 passenger management menu
appears
    {
        // loop terminates if user enters 5
        while (true)
        {
            headerCls();
            managePassengerHeader();

            choice = printMenu(managePassengerMenu, 6); // takes user choice

            if (choice == "1") // if admin wants to add passenger data
            {
                headerCls();
                printSubHeader(headerNames[5]);

                // x is used furthur to check if user added successfully or not
                int x = passengerCountIdx;

                // adds a passenger
```

```
                addPassengerData(passengerTicketStatus, passengerName,
passengerID, passengerIDPassword, passengerCnic, employeeCnic, passengerCountIdx,
employeeCountIdx);


                if (x != passengerCountIdx) // if passsenger added successfully the
file is updated
                {
                   // saves initial values of the ticket of a user
                   ticketStatusPassenger(passengerTicketStatus,
passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx - 1);


                   // saves data on the passenger file
                   passengersNewDataFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                }
             }
             else if (choice == "2") // if admin wants to delete passenger data
             {
                headerCls();
                printSubHeader(headerNames[6]);


                // delete a passenger data


                gotoxy(50, 23);
                cout << "\e[0;37mEnter passengerID:\e[0;32m ";
                string ID = inputs(); // takes passengerId as input
```

```
                    if (userCheck(ID, passengerID, passengerCountIdx) == true) // if passengerID is found
                    {
                        int idx = indexCheck(ID, passengerID, passengerCountIdx); // checks the index of passengerID

                        // displays the data of the passenger to be deleted
                        viewUserData(passengerName, passengerID, passengerIDPassword, passengerCnic, idx, 50, 25);

                        string option = YesNoChoice("Passenger"); // asks the admin for its choice to delete data or not

                        if (option == "1" || option == "Yes") // if admin enters Yes or 1 passenger data is deleted
                        {
                            int delIndex = passengerCountIdx;
                            // deletes the passenger data
                            deleteData(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerCountIdx, idx);

                            // deletes the passenger ticket details
                            deleteTicketData(passengerTicketStatus, trainNo, passengerDepartureCity, passengerArrivalCity, passengerTicketRoute, passengerTicketPrice, idx, delIndex);

                            // updates the file after data deleted successfully
                            passengersDataUpdateFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
```

```
                        printStatement("Passenger data deleted successfully!", 50, 33, " ");
                    }
                    else if (option == "2" || option == "No") // if admin enters No or 2 passenger data is not deleted
                    {
                        printStatement("Passenger data not deleted!", 50, 33, " ");
                    }
                }
                else // if passengerID is not found
                {
                    printStatement("Passenger data not found!", 50, 25, "W");
                }
            }
            else if (choice == "3") // if admin wants to update passenger data
            {
                headerCls();
                printSubHeader(headerNames[7]);

                // update a passenger data
                updateUserData(passengerName, passengerID, passengerIDPassword, passengerCnic, employeeCnic, "Passenger", passengerCountIdx, employeeCountIdx);

                // updates the passenger file after data is updated
                passengersDataUpdateFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
            }
            else if (choice == "4") // if admin wants to view passenger data
```

```
                    {
                        headerCls();
                        printSubHeader(headerNames[8]);

                        // displays all passengers data
                        viewUserDataList(passengerName, passengerID,
passengerIDPassword, passengerCnic, "Passenger", passengerCountIdx);
                    }
                    else if (choice == "5") // if admin wants to search passenger data
enters 5
                    {
                        headerCls();
                        printSubHeader(headerNames[9]);

                        // searches the passenger data by using passenger ID entered by
admin
                        searchUserData(passengerName, passengerID,
passengerIDPassword, passengerCnic, "Passenger", passengerCountIdx);
                    }
                    else if (choice == "6") // if admin wants to go back enters 6
                    {
                        break;
                    }
                    else // if admin enters wrong input
                    {
                        printStatement("Wrong userChoice!...", 50, 30, "W");
                    }
                }
            }
```

```
                else if (choice == "3") // if admin enters 3 train routes management menu appears
                {
                    // loop terminates if admin enters 4
                    while (true)
                    {
                        headerCls();
                        manageTrainsHeader();

                        choice = printMenu(manageTrainMenu, 4); // takes admin's choice as input

                        if (choice == "1") // if admin wants to add train route enters 1
                        {
                            headerCls();
                            printSubHeader(headerNames[10]);

                            int x = trainCountIdx; // temporarily stores trains count

                            // adds a train route data
                            addTrainRoute(trainNo, trainArrivalCity, trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx, cities);

                            if (x != trainCountIdx)
                            {
                                // adds the train data to the file
                                trainsNewDataFile(trainNo, trainRoute, trainArrivalCity, trainDepartureCity, trainTicketPrice, trainCountIdx);
                            }
                        }
```

```
                else if (choice == "2") // if admin wants to delete train route enters 2
                {
                    headerCls();
                    printSubHeader(headerNames[11]);

                    // deletes a train data
                    string trainID = deleteTrainRoute(trainNo, trainArrivalCity,
trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx);

                    if (trainID != " ") // if trainID is found
                    {
                        // if train data is deleted the ticket data corresponding to it also
gets deleted and the passenger file is updated
                        deleteTrainTicketDetails(passengerTicketStatus,
passengerTrainNo, passengerArrivalCity, passengerDepartureCity,
passengerTicketRoute, passengerTicketPrice, passengerCountIdx, trainID);

                        // the train data file is updated after deletion of the train data
                        trainsDataUpdateFile(trainNo, trainRoute, trainArrivalCity,
trainDepartureCity, trainTicketPrice, trainCountIdx);

                        // passenger data file is updated as the tickets data parallel to that
train is deleted
                        passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                    }
                }
                else if (choice == "3") // if admin wants to view trains available
enters 3
                {
```

```
                    headerCls();
                    printSubHeader(headerNames[12]);


                    // displays the trains available
                    viewTrainsAvailable(trainNo, trainArrivalCity,
trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx);
                }
                else if (choice == "4") // if admin wants to go back enters 4
                {
                    break;
                }
                else // if admin enters wrong input
                {
                    printStatement("Wrong userChoice!...", 50, 28, "W");
                }
            }
        }
        else if (choice == "4") // if admin enters 4 ticketing system menu appears
        {
            // loop termintaes if user enters 4
            while (true)
            {
                headerCls();
                manageTicketetingHeader();


                choice = printMenu(manageTicketSystemMenu, 4); // takes admin's
choice as input for submenu


                if (choice == "1") // if admin wants to book a ticket enters 1
```

```
{
    headerCls();
    printSubHeader(headerNames[13]);

    gotoxy(50, 23);
    cout << "\e[0;37mEnter passengerID:\e[0;32m ";
    string ID = inputs(); // takes passengerID as input

    if (userCheck(ID, passengerID, passengerCountIdx) == true) // if passengerID is found
    {
        int index = indexCheck(ID, passengerID, passengerCountIdx); // checks the index of passengerID

        if (trainCountIdx > 0) // if trains are available
        {
            string tempSt = passengerTicketStatus[index]; // saves passenger ticket status temporarily

            // books ticket for a user
            bookTickets(passengerTicketStatus, passengerTrainNo, trainNo, passengerArrivalCity, trainArrivalCity, passengerDepartureCity, trainDepartureCity, passengerTicketRoute, trainRoute, passengerTicketPrice, trainTicketPrice, index, trainCountIdx);

            // updates passengers data file after ticket is booked
            passengersDataUpdateFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
        }
```

```
                else // if trains are not availabe
                {
                    printStatement("Train routes not available!", 50, 25, "W");
                }
            }
            else // if passnegerID not found
            {
                printStatement("PassengerID not found!", 50, 25, "W");
            }
        }
        else if (choice == "2") // if admin wants to cancel ticket enters 2
        {
            headerCls();
            printSubHeader(headerNames[14]);

            gotoxy(50, 23);
            cout << "\e[0;37mEnter passengerID:\e[0;32m ";
            string ID = inputs(); // takes passengerID as input

            if (userCheck(ID, passengerID, passengerCountIdx) == true) // if
passengerID is found
            {
                int index = indexCheck(ID, passengerID, passengerCountIdx); //
checks the index of passengerID

                if (passengerTicketStatus[index] == "Y") // if passenger ticket
data if found
                {
                    // deletes the tickets details of the user
```

```
                        cancelTicket(passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, index);


                        // updates the passenger file after ticket data is deleted

                        passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                    }
                    else // if ticket data not found for the passnger
                    {
                        printStatement("Ticket details not available!", 50, 25, "W");
                    }
                }
                else // if passengerID not found
                {
                    printStatement("PassengerID not found!", 50, 25, "W");
                }
            }
            else if (choice == "3") // if admin wants to view booked ticket details
enters 3
            {
                headerCls();
                printSubHeader(headerNames[15]);


                // displays the tickets booked
                viewBookedTickets(passengerName, passengerCnic,
passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
            }
```

```
                else if (choice == "4") // if admin wants to go back enters 4
                {
                    break;
                }
                else // if admin enters wrong input
                {
                    printStatement("Wrong userChoice!...", 50, 28, "W");
                }
            }
        }
        else if (choice == "5") // if admin wants to view collected revenue enters 5
        {
            headerCls();
            printSubHeader(headerNames[18]);

            // displays the number of tickets in each train and their total collected
revenue
            ticketsDetails(passengerTrainNo, trainNo, passengerTicketPrice,
trainTicketPrice, passengerCountIdx, trainCountIdx);
        }
        else if (choice == "6") // if admin wants to log out
        {
            break;
        }
        else // if admin enters wrong input
        {
            printStatement("Wrong userChoice!...", 50, 30, "W");
        }
    }
```

```
            }
            else // admin sign in failed
            {
                printStatement("Wrong Credentials! Sign In failed!", 50, 25, "W");
            }
        }
        else // if user is employee or a passenger
        {
            bool x = userCheckSignIn(userIDSI, passengerID, employeeID,
passengerCountIdx, employeeCountIdx); // checks if userID exists or not


            if (x == true) // if userID is found in data
            {
                string role = roleCheck(userIDSI); // checks role of the user


                if (role == "Employee") // if role of user is Employee
                {
                    int empIdx = indexCheck(userIDSI, employeeID, employeeCountIdx);
// employee index for ID
                    string empPasswordSI =
userPasswordSignIn(employeeIDPassword[empIdx]); // taskes input for password


                    if (empPasswordSI == employeeIDPassword[empIdx]) // if employee
enters correct password
                    {
                        // loop terminates if employee enters 5
                        while (true)
                        {
                            headerCls();
                            employeeHeader();
```

```
                string choice = printMenu(employeeMenu, 6); // takes employee choice as input

                if (choice == "1") // if employee enters 1 manage passengermenu appears
                {
                    // loop terminates if user enters 6
                    while (true)
                    {
                        headerCls();
                        managePassengerHeader();

                        choice = printMenu(managePassengerMenu, 6); // takes employee's input for submenu

                        if (choice == "1") // if employee wants to add passenger data
                        {
                            headerCls();
                            printSubHeader(headerNames[5]);

                            // x is used furthur to check if user added successfully or not
                            int x = passengerCountIdx;

                            // adds a passenger
                            addPassengerData(passengerTicketStatus, passengerName, passengerID, passengerIDPassword, passengerCnic, employeeCnic, passengerCountIdx, employeeCountIdx);
```

```
                    if (x != passengerCountIdx) // if passsenger added
successfully the file is updated
                    {
                        // saves initial values of the ticket of a user
                        ticketStatusPassenger(passengerTicketStatus,
passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx - 1);


                        // saves data on the passenger file
                        passengersNewDataFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                    }
                }
                else if (choice == "2") // if employee wants to delete passenger
data
                {
                    headerCls();
                    printSubHeader(headerNames[6]);


                    // delete a passenger data


                    gotoxy(50, 23);
                    cout << "\e[0;37mEnter passengerID:\e[0;32m ";
                    string ID = inputs(); // takes passengerId as input


                    if (userCheck(ID, passengerID, passengerCountIdx) == true)
// if passengerID is found
                    {
```

```
                int idx = indexCheck(ID, passengerID,
passengerCountIdx); // checks the index of passengerID


                // displays the data of the passenger to be deleted

                viewUserData(passengerName, passengerID,
passengerIDPassword, passengerCnic, idx, 50, 25);


                string option = YesNoChoice("Passenger"); // asks the
employee for its choice to delete data or not


                if (option == "1" || option == "Yes") // if employee enters
Yes or 1 passenger data is deleted

                {

                    int delIndex = passengerCountIdx;

                    // deletes the passenger data

                    deleteData(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerCountIdx, idx);


                    // deletes the passenger ticket details

                    deleteTicketData(passengerTicketStatus, trainNo,
passengerDepartureCity, passengerArrivalCity, passengerTicketRoute,
passengerTicketPrice, idx, delIndex);


                    // updates the file after data deleted successfully

                    passengersDataUpdateFile(passengerName,
passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus,
passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx);


                    printStatement("Passenger data deleted successfully!",
50, 33, " ");

                }
```

```
                                else if (option == "2" || option == "No") // if employee
enters No or 2 passenger data is not deleted
                                {
                                    printStatement("Passenger data not deleted!", 50, 33, "
");
                                }
                            }
                            else // if passengerID is not found
                            {
                                printStatement("Passenger data not found!", 50, 25, "W");
                            }
                        }
                        else if (choice == "3") // if employee wants to update passenger
data
                        {
                            headerCls();
                            printSubHeader(headerNames[7]);

                            // update a passenger data
                            updateUserData(passengerName, passengerID,
passengerIDPassword, passengerCnic, employeeCnic, "Passenger", passengerCountIdx,
employeeCountIdx);

                            // updates the passneger file after its data is updated
                            passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                        }
                        else if (choice == "4") // if employee wants to view passenger
data
```

```
                {
                    headerCls();
                    printSubHeader(headerNames[8]);

                    // displays all passengers data
                    viewUserDataList(passengerName, passengerID, passengerIDPassword, passengerCnic, "Passenger", passengerCountIdx);
                }
                else if (choice == "5") // if employee wants to search passenger data enters 5
                {
                    headerCls();
                    printSubHeader(headerNames[9]);

                    // searches the passenger data by using passenger ID entered by employee
                    searchUserData(passengerName, passengerID, passengerIDPassword, passengerCnic, "Passenger", passengerCountIdx);
                }
                else if (choice == "6") // if employee wants to go back enters 6
                {
                    break;
                }
                else // if employee enters wrong choice
                {
                    printStatement("Wrong userChoice!...", 50, 30, "W");
                }
            }
        }
```

```
                else if (choice == "2") // if employee enters 2 manage trainmenu appears
                {
                    // loop termintaes if user enters 4
                    while (true)
                    {
                        headerCls();
                        manageTrainsHeader();

                        choice = printMenu(manageTrainMenu, 4); // takes employee's choice as input

                        if (choice == "1") // if employee wants to add train route enters 1
                        {
                            headerCls();
                            printSubHeader(headerNames[10]);

                            int x = trainCountIdx; // temporarily stores trains count

                            // adds a train route data
                            addTrainRoute(trainNo, trainArrivalCity, trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx, cities);

                            if (x != trainCountIdx)
                            {
                                // adds the train data to the file
                                trainsNewDataFile(trainNo, trainRoute, trainArrivalCity, trainDepartureCity, trainTicketPrice, trainCountIdx);
                            }
                        }
```

```
                else if (choice == "2") // if employee wants to delete train route enters 2
                {
                    headerCls();
                    printSubHeader(headerNames[11]);

                    // deletes a train data
                    string trainID = deleteTrainRoute(trainNo, trainArrivalCity, trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx);

                    if (trainID != " ") // if trainID is found
                    {
                        // if train data is deleted the ticket data corresponding to it also gets deleted and the passenger file is updated
                        deleteTrainTicketDetails(passengerTicketStatus, passengerTrainNo, passengerArrivalCity, passengerDepartureCity, passengerTicketRoute, passengerTicketPrice, passengerCountIdx, trainID);

                        // the train data file is updated after deletion of the train data
                        trainsDataUpdateFile(trainNo, trainRoute, trainArrivalCity, trainDepartureCity, trainTicketPrice, trainCountIdx);

                        // passenger data file is updated as the tickets data parallel to that train is deleted
                        passengersDataUpdateFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                    }
                }
```

```
                    else if (choice == "3") // if employee wants to view trains available enters 3
                    {
                        headerCls();
                        printSubHeader(headerNames[12]);

                        // displays the trains available
                        viewTrainsAvailable(trainNo, trainArrivalCity, trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx);
                    }
                    else if (choice == "4") // if employee wants to go back enters 4
                    {
                        break;
                    }
                    else // if employee enters wrong input
                    {
                        printStatement("Wrong userChoice!...", 50, 28, "W");
                    }
                }
            }
            else if (choice == "3") // if employee enters 3 ticketing system menu appears
            {
                while (true)
                {
                    headerCls();
                    manageTicketetingHeader();

                    choice = printMenu(manageTicketSystemMenu, 4); // takes employee's choice as input for submenu
```

```
                    if (choice == "1") // if employee wants to book a ticket for a user enters 1
                    {
                        headerCls();
                        printSubHeader(headerNames[13]);

                        gotoxy(50, 23);
                        cout << "\e[0;37mEnter passengerID:\e[0;32m ";
                        string ID = inputs(); // takes passengerID as input

                        if (userCheck(ID, passengerID, passengerCountIdx) == true) // if passengerID is found
                        {
                            int index = indexCheck(ID, passengerID, passengerCountIdx); // checks the index of passengerID

                            if (trainCountIdx > 0) // if trains are available
                            {
                                string tempSt = passengerTicketStatus[index]; // saves passenger ticket status temporarily

                                // books ticket for a user
                                bookTickets(passengerTicketStatus, passengerTrainNo, trainNo, passengerArrivalCity, trainArrivalCity, passengerDepartureCity, trainDepartureCity, passengerTicketRoute, trainRoute, passengerTicketPrice, trainTicketPrice, index, trainCountIdx);

                                // updates passengers data file after ticket is booked
                                passengersDataUpdateFile(passengerName, passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus,
```

```
passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                                    }
                                    else // if trains are not availabe
                                    {
                                        printStatement("Train routes not available!", 50, 25,
"W");
                                    }
                                }
                                else // if passnegerID not found
                                {
                                    printStatement("PassengerID not found!", 50, 25, "W");
                                }
                            }
                            else if (choice == "2") // if employee wants to cancel ticket
enters 2
                            {
                                headerCls();
                                printSubHeader(headerNames[14]);

                                gotoxy(50, 23);
                                cout << "\e[0;37mEnter passengerID: \e[0;32m";
                                string ID = inputs(); // takes passengerID as input

                                if (userCheck(ID, passengerID, passengerCountIdx) == true)
// if passengerID is found
                                {
                                    int index = indexCheck(ID, passengerID,
passengerCountIdx); // checks the index of passengerID
```

```
                    if (passengerTicketStatus[index] == "Y") // if passenger
ticket data if found
                    {
                        // deletes the tickets details of the user
                        cancelTicket(passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, index);


                        // updates the passenger file after ticket data is deleted
                        passengersDataUpdateFile(passengerName,
passengerID, passengerIDPassword, passengerCnic, passengerTicketStatus,
passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                    }
                    else // if ticket data not found for the passnger
                    {
                        printStatement("Ticket details not available!", 50, 25,
"W");
                    }
                }
                else // if passengerID not found
                {
                    printStatement("PassengerID not found!", 50, 25, "W");
                }
            }
            else if (choice == "3") // if employee wants to view tickets
booked enters 3
            {
                headerCls();
                printSubHeader(headerNames[15]);
```

```
                    // displays the tickets booked
                    viewBookedTickets(passengerName, passengerCnic,
passengerTicketStatus, passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, passengerCountIdx);
                }
                else if (choice == "4") // if employee wants to go back enters 4
                {
                    break;
                }
                else // if employee enters wrong input
                {
                    printStatement("Wrong userChoice!...", 50, 28, "W");
                }
            }
        }
        else if (choice == "4") // if employee enters 4 employee data appears
        {
            headerCls();
            printSubHeader(headerNames[16]);

            // displays the data of the employee signed in
            viewUserData(employeeName, employeeID,
employeeIDPassword, employeeCnic, empIdx, 50, 24);
            pressAnyKey(50, 29);
        }
        else if (choice == "5") // if employee wants to change password
enters 5
        {
            headerCls();
            printSubHeader(headerNames[19]);
```

```
                        // changes the password
                        string newPassword = changePassword(employeeIDPassword,
empIdx);

                        if (newPassword != " ")
                        {
                            // employee file is updated
                            employeesDataUpdateFile(employeeName, employeeID,
employeeIDPassword, employeeCnic, employeeCountIdx);
                        }
                    }
                    else if (choice == "6") // if employee wants to log out enters 6
                    {
                        break;
                    }
                    else // if employee enters wrong input
                    {
                        printStatement("Wrong userChoice!...", 50, 30, "W");
                    }
                }
            }
            else // if user enters wrong credentials
            {
                printStatement("Wrong Credentials! Sign In failed!", 50, 25, "W");
            }
        }
        else if (role == "Passenger") // if role of user is Passenger
        {
```

```
                int pasgIdx = indexCheck(userIDSI, passengerID, passengerCountIdx);
// user index check

                string pasgPasswordSI =
userPasswordSignIn(passengerIDPassword[pasgIdx]); // taskes input for password


                if (pasgPasswordSI == passengerIDPassword[pasgIdx]) // if user enters
correct password

                {

                   while (true)

                   {

                      headerCls();

                      passengerHeader();


                      string choice = printMenu(passengerMenu, 7); // takes user choice as
input


                      if (choice == "1") // if user wants to book ticket enters 1

                      {

                         headerCls();

                         printSubHeader(headerNames[13]);


                         if (trainCountIdx > 0) // if train routes available

                         {

                            // allows user to book ticket

                            bookTickets(passengerTicketStatus, passengerTrainNo, trainNo,
passengerArrivalCity, trainArrivalCity, passengerDepartureCity, trainDepartureCity,
passengerTicketRoute, trainRoute, passengerTicketPrice, trainTicketPrice, pasgIdx,
trainCountIdx);


                            // updates user file after booking ticket
```

```
                    passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);

                }

                else // if train routes not available

                {

                    printStatement("Train routes not available!", 50, 25, "W");

                }

            }

            else if (choice == "2") // if user wants to cancel ticket enters 2

            {

                headerCls();

                printSubHeader(headerNames[14]);


                if (passengerTicketStatus[pasgIdx] == "Y") // if ticket details are
available allows user to cancel ticket

                {

                    // deletes user ticket details

                    cancelTicket(passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, pasgIdx);


                    // updates user data file

                    passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);

                }

                else // if ticket details not available

                {
```

```
                printStatement("Ticket details not available!", 50, 25, "W");
            }
        }
        else if (choice == "3") // if user wants to view timetable of the trains enters 3
        {
            headerCls();
            printSubHeader(headerNames[12]);

            // displays the trains available
            viewTrainsAvailable(trainNo, trainArrivalCity, trainDepartureCity, trainRoute, trainTicketPrice, trainCountIdx);
        }
        else if (choice == "4") // if user wants to view its own data enters 4
        {
            headerCls();
            printSubHeader(headerNames[16]);

            // displays user data
            viewUserData(passengerName, passengerID, passengerIDPassword, passengerCnic, pasgIdx, 50, 24);
            pressAnyKey(50, 29);
        }
        else if (choice == "5") // if user wants to view his bokked ticket details enters 5
        {
            headerCls();
            printSubHeader(headerNames[17]);

            // displays the ticket details of the passenger
```

```
                viewTicketDetails(passengerTrainNo, passengerTicketRoute,
passengerArrivalCity, passengerDepartureCity, passengerTicketPrice, pasgIdx);
                pressAnyKey(50, 28);
            }
            else if (choice == "6") // if user wants to change password enters 6
            {
                headerCls();
                printSubHeader(headerNames[19]);

                // changes the password
                string newPassword = changePassword(passengerIDPassword,
pasgIdx);

                if (newPassword != " ")
                {
                    // updates the passneger file after its data is updated
                    passengersDataUpdateFile(passengerName, passengerID,
passengerIDPassword, passengerCnic, passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, passengerCountIdx);
                }
            }
            else if (choice == "7") // if user wants to log out enters 7
            {
                break;
            }
            else // if user enters wrong input
            {
                printStatement("Wrong userChoice!...", 50, 31, "W");
            }
```

```
                    }
                }
                else // if passenger enters wrong password
                {
                    printStatement("Wrong Credentials! Sign In failed!", 50, 25, "W");
                }
            }
        }
        else // if user enters wrong userID
        {
            printStatement("UserID not found! Sign in failed!", 50, 24, "W");
        }
    }
}
else if (option == "3") // if user enters 3 exits the program
{
    return 0;
}
else // if user enter wrong input
{
    printStatement("Wrong userChoice!...", 50, 27, "W");
}
}
}

// header functions
void header()
{
```

```
    gotoxy(10, 2);

    cout << "\e[1;33m    ______       _     _       _                ______        _  _" << endl;

    gotoxy(10, 3);

    cout << "    | ___ \\     | |   (_)     | |               | ___ \\      (_)| |" << endl;

    gotoxy(10, 4);

    cout << "    | |_/ /__ _ | | __ _  ___ | |_  __ _  _ __   | |_/ / __ _  _ | |__      __ __ _  _   _
___" << endl;

    gotoxy(10, 5);

    cout << "    |  __// _` || |/ /| |/ __|| __|/ _` || '_ \\  |    / / _` || || |\\ \\ /\\ / // _` || | | |/ __|" <<
endl;

    gotoxy(10, 6);

    cout << "    | |  | (_| ||   < | |\\__ \\| |_| (_| || | | | | |\\ \\| (_| || || | \\ V  V /| (_| || |_| |\\__ \\" <<
endl;

    gotoxy(10, 7);

    cout << "    \\_|   \\__,_||_|\\_\\|_||___/ \\___|\\___,_||_| |_| \\_| \\_|\\___,_||_||_|  \\_/\\_/  \\___,_|
\\___, ||___/" << endl;

    gotoxy(10, 8);

    cout << "\e[1;31m___  ___                                                  _     _____
\e[1;33m__/  | \e[1;31m_" << endl;

    gotoxy(10, 9);

    cout << "|  \\/  |                                                | |   /  ___|     \e[1;33m|____/
\e[1;31m| |" << endl;

    gotoxy(10, 10);

    cout << "| .  . | __ _  _ __    __ _   __ _   ___  _ __ ___    ___  _ __  | |_  \\ `--.  _   _  ___
| |_  ___  _ __ ___" << endl;

    gotoxy(10, 11);

    cout << "| |\\/| | / _` || '_ \\  / _` | / _` | / _ \\| '_ ` _ \\  / _ \\| '_ \\ | __|  `--. \\| | |/ __|| __|/ _
\\| '_ ` _ \\" << endl;

    gotoxy(10, 12);

    cout << "| |  | || (_| || | | || (_| || (_| ||  __/| | | | | ||  __/| | | || |_  /\\__/ /| |_| |\\__ \\| |_|  __/| | | | |
|" << endl;
```

```
    gotoxy(10, 13);
    cout << "\\_|  |_/ \\__,_||_| |_| \\__,_| \\__, | \\____||_| |_| |_| \\____||_| |_| \\__| \\_____/  \\__,
||____/ \\__|\\____||_| |_| |_|" << endl;
    gotoxy(10, 14);
    cout << "                          __/ |                                __/ |" << endl;
    gotoxy(10, 15);
    cout << "                         |___/                                |___/" << endl;
}
void startHeader()
{
    gotoxy(10, 2);
    Sleep(70);
    cout << "\e[1;33m    _______       _     _      _               _______        _  _" << endl;
    gotoxy(10, 3);
    Sleep(70);
    cout << "    | ___ \\     | |   (_)     | |              | ___ \\      (_)| |" << endl;
    gotoxy(10, 4);
    Sleep(70);
    cout << "    | |_/ /__ _ | | __ _  ___ | |_  __ _  _ __   | |_/ / __ _  _ | |__      __ __ _  _   _
___" << endl;
    gotoxy(10, 5);
    Sleep(70);
    cout << "    |  __// _` || |/ /| |/ __|| __|/ _` || '_ \\  |    / / _` || || |\\ \\ /\\ / // _` || | | |/ __|" <<
endl;
    gotoxy(10, 6);
    Sleep(70);
    cout << "    | |  | (_| ||   < | |\\__ \\| |_| (_| || | | | | |\\ \\| (_| || || | \\ V  V /| (_| || |_| |\\__ \\" <<
endl;
    gotoxy(10, 7);
    Sleep(70);
```

```
    cout << "    \\_|   \\__,_||_|\\_\\|_||___/ \\__|\\__,_||_| |_| \\_| \\_|\\__,_||_||_|  \\_/\\_/  \\__,_|
\\__, ||___/" << endl;

    gotoxy(10, 8);

    Sleep(70);

    cout << "\e[1;31m___  ___                                                  _     _____
\e[1;33m__/  | \e[1;31m_" << endl;

    gotoxy(10, 9);

    Sleep(70);

    cout << "|  \\/  |                                                  | |   /  ___|     \e[1;33m|____/
\e[1;31m| |" << endl;

    gotoxy(10, 10);

    Sleep(70);

    cout << "| .  . | __ _  _ __    __ _   __ _   ___  _ __ ___    ___  _ __  | |_  \\ `--.  _   _  ___
| |_  ___  _ __ ___" << endl;

    gotoxy(10, 11);

    Sleep(70);

    cout << "| |\\/| | / _` || '_ \\  / _` | / _` | / _ \\| '_ ` _ \\  / _ \\| '_ \\ | __|  `--. \\| | | |/ __|| __|/ _
\\| '_ ` _ \\" << endl;

    gotoxy(10, 12);

    Sleep(70);

    cout << "| |  | || (_| || | | || (_| || (_| ||  __/| | | | | ||  __/| | | || |_  /\\__/ /| |_| |\\__ \\| |_|  __/| | | | |
|" << endl;

    gotoxy(10, 13);

    Sleep(70);

    cout << "\\_|  |_/ \\__,_||_| |_| \\__,_| \\__, | \\____||_| |_| |_| \\____||_| |_| \\___| \\_____/  \\__,
||___/ \\__|\\____||_| |_| |_|" << endl;

    gotoxy(10, 14);

    Sleep(70);

    cout << "                          __/ |                                  __/ |" << endl;

    gotoxy(10, 15);

    Sleep(70);
```

```
    cout << "                    |___/                              |___/" << endl;
}
void startMenuHeader()
{
    gotoxy(50, 19);
    cout << "\e[0;34m*************************" << endl;
    gotoxy(50, 20);
    cout << "*       \e[1;33mMain Menu\e[0;34m       *" << endl;
    gotoxy(50, 21);
    cout << "*************************\e[0;37m" << endl;
}
void signUpHeader()
{
    gotoxy(50, 19);
    cout << "\e[0;34m**********************" << endl;
    gotoxy(50, 20);
    cout << "*      \e[1;33mSign Up\e[0;34m      *" << endl;
    gotoxy(50, 21);
    cout << "**********************\e[0;37m" << endl;
}
void signInHeader()
{
    gotoxy(50, 19);
    cout << "\e[0;34m**********************" << endl;
    gotoxy(50, 20);
    cout << "*      \e[1;33mSign In\e[0;34m      *" << endl;
    gotoxy(50, 21);
    cout << "**********************\e[0;37m" << endl;
```

```
}
void adminHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mAdmin Menu\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "************************\e[0;37m" << endl;
}
void manageEmployeeHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m************************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mManage Employee Data\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "************************************\e[0;37m" << endl;
}
void managePassengerHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m************************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mManage Passenger Data\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "************************************\e[0;37m" << endl;
}
```

```
void manageTrainsHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m**********************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mManage Train Routes\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "**********************************\e[0;37m" << endl;
}
void manageTicketetingHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m*******************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mTicketing System\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "*******************************\e[0;37m" << endl;
}
void employeeHeader()
{
   gotoxy(50, 19);
   cout << "\e[0;34m****************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mEmployee Menu\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "****************************\e[0;37m" << endl;
}
void passengerHeader()
```

```
{
   gotoxy(50, 19);
   cout << "\e[0;34m*****************************" << endl;
   gotoxy(50, 20);
   cout << "*      \e[1;33mPassenger Menu\e[0;34m      *" << endl;
   gotoxy(50, 21);
   cout << "*****************************\e[0;37m" << endl;
}
void printSubHeader(string menuHeader)
{
   gotoxy(50, 21);
   cout << "\e[0;34m*      \e[1;33m" << menuHeader << "\e[0;34m      *\e[0;37m" <<
endl;
}

// instructions for users
void noteUserName()
{
   gotoxy(10, 19);
   cout << "\e[1;31m            Instructions" << endl;
   gotoxy(10, 21);
   cout << "\e[0;31m-->\e[0;33mFirst letter must be capital." << endl;
   gotoxy(10, 22);
   cout << "\e[0;31m-->\e[0;33mNumbers and special characters are" << endl;
   gotoxy(10, 23);
   cout << "   \e[0;33mnot allowed to enter.  " << endl;
}
void noteuserIDpassenger()
{
```

```
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mSpaces not allowed." << endl;
    gotoxy(10, 22);
    cout << "\e[0;31m-->\e[0;33m\"emp\". should not be the first" << endl;
    gotoxy(10, 23);
    cout << "   \e[0;33mpart of userID." << endl;
    gotoxy(10, 24);
    cout << "\e[0;31m-->\e[0;33mUserID cannot be \"admin\"." << endl;
    gotoxy(10, 25);
    cout << "\e[0;31m-->\e[0;33mUserID must consist of 6-12" << endl;
    gotoxy(10, 26);
    cout << "   \e[0;33mcharacters!" << endl;
    gotoxy(10, 27);
    cout << "\e[0;31m-->\e[0;33mAlready existing userID cannot" << endl;
    gotoxy(10, 28);
    cout << "   \e[0;33mbe used" << endl;
}
void noteSUpassword()
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mSpaces not allowed." << endl;
    gotoxy(10, 22);
    cout << "\e[0;31m-->\e[0;33mMust consist of 6-12 characters." << endl;
    gotoxy(10, 23);
```

```
    cout << "\e[0;31m-->\e[0;33mAtleast 1 letter and 1 number" << endl;
    gotoxy(10, 24);
    cout << "   \e[0;33mshould be entered." << endl;
    gotoxy(10, 25);
    cout << "\e[0;31m-->\e[0;33mSpecial Characters can be used. " << endl;
}
void noteSUcnic()
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mSpaces not allowed." << endl;
    gotoxy(10, 22);
    cout << "\e[0;31m-->\e[0;33mCNIC must consist of 13 numbers." << endl;
    gotoxy(10, 23);
    cout << "\e[0;31m-->\e[0;33mEnter your own CNIC number." << endl;
}
void eraseInstruction()
{
    gotoxy(10, 19);
    cout << "                                " << endl;
    gotoxy(10, 21);
    cout << "                                " << endl;
    gotoxy(10, 22);
    cout << "                                " << endl;
    gotoxy(10, 23);
    cout << "                                " << endl;
    gotoxy(10, 24);
```

```
    cout << "                                " << endl;
    gotoxy(10, 25);
    cout << "                                " << endl;
    gotoxy(10, 26);
    cout << "                                " << endl;
    gotoxy(10, 27);
    cout << "                                " << endl;
    gotoxy(10, 28);
    cout << "                                " << endl;
}
void forgottenIDorPassword()
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mIf you have forgotten your ID" << endl;
    gotoxy(10, 22);
    cout << "   or password enter \"IDRECOVERY\"" << endl;
    gotoxy(10, 23);
    cout << "   in main menu." << endl;
}
void noteAddEmployee()
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mSpaces not allowed." << endl;
    gotoxy(10, 22);
```

```
    cout << "\e[0;31m-->\e[0;33m\"emp\". should be the first" << endl;
    gotoxy(10, 23);
    cout << "   \e[0;33mpart of employeeID." << endl;
    gotoxy(10, 24);
    cout << "\e[0;31m-->\e[0;33mEmployeeID cannot be \"admin\"." << endl;
    gotoxy(10, 25);
    cout << "\e[0;31m-->\e[0;33mEmployeeID must consist of minimum" << endl;
    gotoxy(10, 26);
    cout << "   \e[0;33mof 6 characters!" << endl;
}
void noteAddTrain()
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mSpaces not allowed." << endl;
    gotoxy(10, 22);
    cout << "\e[0;31m-->\e[0;33mOnly '-' can be used in trainID." << endl;
    gotoxy(10, 23);
    cout << "\e[0;31m-->\e[0;33mAlready existing trainID cannot be" << endl;
    gotoxy(10, 24);
    cout << "\e[0;31m-->\e[0;33madded again!" << endl;
}
void noteDepartureCity(string cities[])
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
```

```
    cout << "\e[0;31m-->\e[0;33mEnter one of the following:\e[0;37m" << endl;
    int y = 22;
    for (int i = 0; i < 15; i++)
    {
        gotoxy(10, y);
        cout << "   " << cities[i] << "          ";
        y = y + 1;
    }
}
void noteArrivalCity(string departurecity, string cities[])
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mArrival and departure city " << endl;
    gotoxy(10, 22);
    cout << "\e[0;31m   \e[0;33mcannot be same.Enter 1 of the " << endl;
    gotoxy(10, 23);
    cout << "\e[0;31m   \e[0;33mfollowing: \e[0;37m" << endl;
    int y = 24;
    for (int i = 0; i < 15; i++)
    {
        if (departurecity == cities[i])
        {
            continue;
        }
        gotoxy(10, y);
        cout << "   " << cities[i] << "          ";
```

```
        y = y + 1;
    }
}
void noteRoutesavail(string trainNo[], string trainRoute[], int trainCountIdx)
{
    gotoxy(10, 19);
    cout << "\e[1;31m            Instructions" << endl;
    gotoxy(10, 21);
    cout << "\e[0;31m-->\e[0;33mChoose one of the following:\e[0;37m" << endl;
    gotoxy(10, 23);
    cout << "\e[0;33m" << setfill(' ') << setw(7) << left << "Sr.No" << setfill(' ') << setw(10) << left << "TrainNo" << setfill(' ') << setw(20) << left << "Route\e[0;32m" << endl;
    if (trainCountIdx > 0)
    {
        int y = 24;

        for (int idx = 0; idx < trainCountIdx; idx++)
        {
            gotoxy(10, y);
            cout << setfill(' ') << setw(7) << left << idx + 1 << setfill(' ') << setw(10) << left << trainNo[idx] << setfill(' ') << setw(20) << left << trainRoute[idx] << endl;
            y = y + 1;
        }
        gotoxy(10, y + 1);
        cout << "\e[0;31m-->\e[0;33mEnter \"esc\" to exit!\e[0;37m" << endl;
    }
}
```

```
// ptinting menu funciton
string printMenu(string menu[], int size)
{
    int y = 23;
    for (int i = 0; i < size; i++)
    {
        gotoxy(50, y);
        cout << i + 1 << "." << menu[i];
        y++;
    }

    gotoxy(50, y + 1);
    cout << "Enter option...\e[0;32m";

    string option = inputs();

    return option;
}

// user signup functions
string userNameSignUp(int y)
{
    noteUserName(); //  prints the instructions for user name

    string userName = " ";

    for (int i = 0; i < 3; i++)
    {
```

```
        if (i == 0) // if user enters name for the first time
        {
            space(62, y);
            gotoxy(50, y);
            cout << "\e[0;37mEnter Name:\e[0;32m ";
            userName = inputs();
        }
        else // if user enters name in wrong format
        {
            space(84, y);
            gotoxy(50, y);
            cout << "\e[0;31mInvalid Name format! \e[0;37mEnter again:\e[0;32m ";
            userName = inputs();
        }
        if (userNameValidationCheck(userName) == true) // checks the validation of the
user Name
        {
            break;
        }
    }

    eraseInstruction(); // erases the instructions

    return userName;
}
bool userNameValidationCheck(string userName)
{
    bool check = true;
```

```
    if (userName[0] >= 'A' && userName[0] <= 'Z')
    {
        int i = 0;
        while (userName[i] != '\0')
        {
            if ((userName[i] < 'A' || userName[i] > 'Z') && (userName[i] < 'a' || userName[i] > 'z') && userName[i] != ' ')
            {
                check = false;
                break;
            }
            i++;
        }
    }
    else
    {
        check = false;
    }

    return check;
}
string userIDSignup(string passengerID[], int passengerCountIdx)
{
    noteuserIDpassenger();

    string ID;

    for (int i = 0; i < 3; i++)
    {
```

```
        ID = userIDInput(i, 24);
        if (userIDCheckSignup(ID, passengerID, passengerCountIdx) == false)
        {
            break;
        }
    }

    eraseInstruction();

    return ID;
}
bool userIDCheckSignup(string ID, string passengerID[], int passengerCountIdx)
{
    bool check = false;

    if (ID.length() >= 6 && ID.length() <= 12)
    {
        int i = 0;
        while (ID[i] != '\0')
        {
            if (ID[i] == ' ' || ID[i] == ';')
            {
                check = true;
            }
            i++;
        }
        if (check == false)
        {
```

```
            if ((ID[0] != 'e' || ID[1] != 'm' || ID[2] != 'p') && ID != "admin")
            {
                check = userCheck(ID, passengerID, passengerCountIdx);
            }
            else
            {
                check = true;
            }
        }
    }
    else
    {
        check = true;
    }

    return check;
}
string userPasswordSignup(int y)
{
    noteSUpassword();

    string password;

    for (int i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            gotoxy(50, y);
```

```
            cout << "\e[0;37mEnter password(6 to 12 characters):\e[0;32m ";
            password = inputs();
        }
        else
        {
            space(86, y);
            gotoxy(50, y);
            cout << "\e[0;31mInvalid Pasword format! \e[0;37mEnter again:\e[0;32m ";
            password = inputs();
        }
        if (passwordValidationCheckSignup(password) == true)
        {
            break;
        }
    }

    eraseInstruction();

    return password;
}
bool passwordValidationCheckSignup(string password)
{
    int charCount = 0; // counts number of characters in the password
    int numCount = 0;  // counts number of numbers in the password

    if (password.length() >= 6 && password.length() <= 12)
    {
        int a = 0;
```

```
        while (!password[a] == '\0' || password[a] == ' ')
        {
            if (password[a] >= '0' && password[a] <= '9')
            {
                numCount++;
            }
            else if ((password[a] >= 'a' && password[a] <= 'z') || (password[a] >= 'A' &&
password[a] <= 'Z'))
            {
                charCount++;
            }
            else if (password[a] == ' ' || password[a] == ';')
            {
                numCount = 0;
                break;
            }

            a++;
        }
        if (numCount > 0 && charCount > 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
```

```
    {
        return false;
    }
}
string userCnicSignup(string passengerCnic[], string employeeCnic[], int
passengerCountIdx, int employeeCountIdx, int y)
{
    noteSUcnic();

    string cnic;

    for (int i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            gotoxy(50, y);
            cout << "\e[0;37mEnter Cnic:\e[0;32m ";
            cnic = inputs();
        }
        else
        {
            space(77, y);
            gotoxy(50, y);
            cout << "\e[0;31mInvalid Cnic! \e[0;37mEnter again:\e[0;32m ";
            cnic = inputs();
        }
        if (userCnicValidationSignup(cnic, passengerCnic, employeeCnic,
passengerCountIdx, employeeCountIdx) == true)
        {
```

```
            break;
        }
    }

    eraseInstruction();

    return cnic;
}
bool userCnicValidationSignup(string cnic, string passengerCnic[], string
employeeCnic[], int passengerCountIdx, int employeeCountIdx)
{
    bool cnicCheck = true;
    if (cnic.length() != 13)
    {
        cnicCheck = false;
    }
    else
    {
        int b = 0;
        while (cnic[b] != '\0')
        {
            if (cnic[b] < '0' || cnic[b] > '9')
            {
                cnicCheck = false;
                break;
            }
            b++;
        }
        if (cnicCheck == true)
```

```
        {
            for (int c = 0; c < passengerCountIdx; c++)
            {
                if (cnic == passengerCnic[c])
                {
                    cnicCheck = false;
                }
            }
            if (cnicCheck == true)
            {
                for (int c = 0; c < employeeCountIdx; c++)
                {
                    if (cnic == employeeCnic[c])
                    {
                        cnicCheck = false;
                        break;
                    }
                }
            }
        }
    }


    return cnicCheck;
}
void saveSUInformation(string name, string userName[], string ID, string userID[], string password, string idPassword[], string cnic, string userCnic[], int &userCountIdx)
{
    int idxSU = userCountIdx;
    userName[idxSU] = name;
```

```
    userID[idxSU] = ID;
    idPassword[idxSU] = password;
    userCnic[idxSU] = cnic;
    userCountIdx++;
}
void ticketStatusPassenger(string passengerTicketStatus[], string passengerTrainNo[],
string passengerTicketRoute[], string passengerArrivalCity[], string
passengerDepartureCity[], int passengerTicketPrice[], int index)
{
    passengerTicketStatus[index] = "N.A";
    passengerTrainNo[index] = "N.A";
    passengerTicketRoute[index] = "N.A";
    passengerArrivalCity[index] = "N.A";
    passengerDepartureCity[index] = "N.A";
    passengerTicketPrice[index] = 0.0;
}

// user sign in functions
string userIDSignIn(string passengerID[], string employeeID[], int passengerCountIdx,
int employeeCountIDx)
{
    string ID;

    for (int i = 0; i < 3; i++)
    {
        ID = userIDInput(i, 23);
        if (userCheckSignIn(ID, passengerID, employeeID, passengerCountIdx,
employeeCountIDx) == true || ID == "admin")
        {
            break;
```

```
        }
    }

    return ID;
}
bool userCheckSignIn(string ID, string passengerID[], string employeeID[], int
passengerCountIdx, int employeeCountIdx)
{
    bool check = false;

    if (ID[0] == 'e' && ID[1] == 'm' && ID[2] == 'p')
    {
        check = userCheck(ID, employeeID, employeeCountIdx);
    }
    else
    {
        check = userCheck(ID, passengerID, passengerCountIdx);
    }

    return check;
}
string adminPasswordCheck()
{
    string password;
    for (int i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            gotoxy(50, 24);
```

```
            cout << "\e[0;37mEnter password:\e[0;32m ";
            password = inputs();
        }
        else
        {
            space(76, 24);
            gotoxy(50, 24);
            cout << "\e[0;31mInvalid Password! \e[0;37mEnter again:\e[0;32m ";
            password = inputs();
        }
        if (password == "admin")
        {
            break;
        }
    }
    return password;
}
bool userCheck(string ID, string userID[], int userCountIdx)
{
    bool x = false;

    for (int idx = 0; idx < userCountIdx; idx++)
    {
        if (ID == userID[idx])
        {
            x = true;
            break;
        }
```

```
    }


    return x;
}
string roleCheck(string ID)
{
    if (ID[0] == 'e' && ID[1] == 'm' || ID[2] == 'p')
    {
        return "Employee";
    }
    else
    {
        return "Passenger";
    }
}
int indexCheck(string ID, string userID[], int userCountIdx)
{
    int index;
    for (int idx = 0; idx < userCountIdx; idx++)
    {
        if (ID == userID[idx])
        {
            index = idx;
            break;
        }
    }
    return index;
}
```

```
string userPasswordSignIn(string idPassword)
{
    string password;

    for (int i = 0; i < 3; i++)
    {
        if (i == 0)
        {
            gotoxy(50, 24);
            cout << "\e[0;37mEnter Password:\e[0;32m ";
            password = inputs();
        }
        else
        {
            space(80, 24);
            gotoxy(50, 24);
            cout << "\e[0;31mInvalid Password! \e[0;37mEnter again:\e[0;32m ";
            password = inputs();
        }
        if (password == idPassword)
        {
            break;
        }
    }

    return password;
}
```

```
// add employee functions

void addEmployeeData(string employeeName[], string employeeID[], string
employeeIDPassword[], string employeeCnic[], string passengerCnic[], int
&employeeCountIdx, int passengerCountIdx)

{

   if (employeeCountIdx == 100)

   {

      printStatement("Sorry Employee cannot be added!", 50, 24, "W");

   }

   else

   {

      string empNamein = userNameSignUp(23);                  // name of employee

      bool empNameCheck = userNameValidationCheck(empNamein); // checks if name
entered is valid


      if (empNameCheck == true)

      {

         string empIdin = empUserIDInput(employeeID, employeeCountIdx);           //
username of employee ID

         bool empIDCheck = employeeIDCheck(empIdin, employeeID,
employeeCountIdx); // if employee ID already exists return true otherwise false


         if (empIDCheck == true) // if employee ID already exists or wrong input

         {

            printStatement("Invalid employee IDs. Failed to add employee!", 50, 25, "W");

         }

         else // if employee ID does not exist

         {

            string empPasswordin = userPasswordSignup(25);                  // password for
employee ID
```

```
            bool passValidation = passwordValidationCheckSignup(empPasswordin); // if
password is valid returns true


            if (passValidation == true) // if admin uses correct password format for
employee ID

            {

               string empCnicin = userCnicSignup(passengerCnic, employeeCnic,
passengerCountIdx, employeeCountIdx, 26);              // cnic of employee

               bool cnicCheck = userCnicValidationSignup(empCnicin, passengerCnic,
employeeCnic, passengerCountIdx, employeeCountIdx); // checks if employee cnic is
valid


               if (cnicCheck == true) // if employee cnic is correct

               {

                  // saving info of the emoloyee

                  saveSUInformation(empNamein, employeeName, empIdin, employeeID,
empPasswordin, employeeIDPassword, empCnicin, employeeCnic, employeeCountIdx);


                  printStatement("Employee added successfully!", 50, 27, " ");

                  // saving data in file

                  employeesNewDataFile(employeeName, employeeID,
employeeIDPassword, employeeCnic, employeeCountIdx);

               }

               else // if employee cnic was not correct

               {

                  printStatement("Invalid employee Cnic. Failed to add employee!", 50, 27,
"W");

               }

            }

            else // if admin enters wrong format for password for employee ids

            {
```

```
                printStatement("Invalid Passwords. Failed to add employee!", 50, 26, "W");
            }
        }
    }
    else
    {
        printStatement("Invalid Name format. Failed to add employee!", 50, 24, "W");
    }
  }
}
string empUserIDInput(string employeeID[], int employeeCountIdx)
{
  noteAddEmployee();

  string userName;

  for (int i = 0; i < 3; i++)
  {
    userName = userIDInput(i, 24);
    if (employeeIDCheck(userName, employeeID, employeeCountIdx) == false)
    {
      break;
    }
  }

  eraseInstruction();

  return userName;
```

```
}
bool employeeIDCheck(string empIDin, string employeeID[], int employeeCountIdx)
{
   bool check = false;


   if (empIDin.length() >= 6)
   {
      int i = 0;
      while (empIDin[i] != '\0')
      {
         if (empIDin[i] == ' ' || empIDin[i] == ';')
         {
            check = true;
         }
         i++;
      }
      if (check == false)
      {
         if ((empIDin[0] == 'e' && empIDin[1] == 'm' && empIDin[2] == 'p') &&
empIDin != "admin")
         {
            check = userCheck(empIDin, employeeID, employeeCountIdx);
         }
         else
         {
            check = true;
         }
      }
   }
```

```
    else
    {
        check = true;
    }

    return check;
}

// add passenger ID functions
void addPassengerData(string passengerTicketStatus[], string passengerName[], string passengerID[], string passengerIDPassword[], string passengerCnic[], string employeeCnic[], int &passengerCountIdx, int employeeCountIdx)
{
    if (passengerCountIdx == 1000) // if username space not available
    {
        printStatement("Sorry, PassengerID cannot be added up!", 50, 24, "W");
    }
    else // if username space is available
    {
        string passengerNameSU = userNameSignUp(23);                    // name of user
        bool passengerNameCheck = userNameValidationCheck(passengerNameSU); // checks if Name enterd by the user is in correct format or not

        if (passengerNameCheck == true)
        {
            string userIDSU = userIDSignup(passengerID, passengerCountIdx);
// username for signing up
            bool userNameCheckSU = userIDCheckSignup(userIDSU, passengerID, passengerCountIdx); // if username already exists return true otherwise false
```

```
        if (userNameCheckSU == true) // if username already exists or wrong input
        {
            printStatement("Invalid UserID. Failed adding PassengerID!", 50, 25, "W");
        }
        else // if username does not exist
        {
            string userPasswordSU = userPasswordSignup(25);                 // password for signing up
            bool validation = passwordValidationCheckSignup(userPasswordSU); // if password is valid returns true

            if (validation == true) // if passenger uses correct password format
            {
                string userCnicSU = userCnicSignup(passengerCnic, employeeCnic, passengerCountIdx, employeeCountIdx, 26);              // cnic for signing up
                bool cnicCheck = userCnicValidationSignup(userCnicSU, passengerCnic, employeeCnic, passengerCountIdx, employeeCountIdx); // checks if cnic is valid

                if (cnicCheck == true) // if cnic is correct
                {
                    // saving info for sign up
                    saveSUInformation(passengerNameSU, passengerName, userIDSU, passengerID, userPasswordSU, passengerIDPassword, userCnicSU, passengerCnic, passengerCountIdx);

                    printStatement("PassengerID added successfully!", 50, 27, " ");
                }
                else // if cnic format is not correct
                {
                    printStatement("Invalid Cnic. Failed adding PassengerID!", 50, 27, "W");
```

```
                }
            }
            else // if user uses wrong format for password
            {
                printStatement("Invalid Password format. Failed adding PassengerID!", 50, 26, "W");
            }
        }
    }
    else
    {
        printStatement("Invalid Name format. Failed adding PassengerID!", 50, 24, "W");
    }
  }
}

// user data fucntions
void deleteUserData(string userName[], string userID[], string userIDPassword[], string userCnic[], string uCode, int &userCountIdx)
{
  gotoxy(50, 23);
  cout << "Enter " << uCode << "ID:\e[0;32m ";
  string ID = inputs();

  if (userCheck(ID, userID, userCountIdx) == true)
  {
    int idx = indexCheck(ID, userID, userCountIdx);

    viewUserData(userName, userID, userIDPassword, userCnic, idx, 50, 25);
```

```
        string option = YesNoChoice(uCode);

        if (option == "1" || option == "Yes")
        {
            deleteData(userName, userID, userIDPassword, userCnic, userCountIdx, idx);
            printStatement(uCode + " data deleted successfully!", 50, 33, " ");
        }
        else if (option == "2" || option == "No")
        {
            printStatement(uCode + " data not deleted!", 50, 33, " ");
        }
    }
    else
    {
        gotoxy(50, 25);
        cout << "\e[0;31m" << uCode << " data not found!";
        pressAnyKey(50, 26);
    }
}
string YesNoChoice(string uCode)
{
    gotoxy(50, 30);
    cout << "\e[0;34mAre you sure you want to delete " << uCode << " data?";
    gotoxy(50, 31);
    cout << "      1.Yes          2.No";
    string option;
    while (true)
    {
```

```
        space(64, 32);
        gotoxy(50, 32);
        cout << "\e[0;37mEnter option:\e[0;32m ";
        option = inputs();
        if (option == "Yes" || option == "No" || option == "1" || option == "2")
        {
            break;
        }
    }

    return option;
}
void deleteData(string userName[], string userID[], string userIDPassword[], string
userCnic[], int &userCountIdx, int index)
{
    for (int j = index; j < userCountIdx; j++)
    {
        userName[j] = userName[j + 1];
        userID[j] = userID[j + 1];
        userIDPassword[j] = userIDPassword[j + 1];
        userCnic[j] = userCnic[j + 1];
    }

    userCountIdx = userCountIdx - 1;
}
void deleteTicketData(string passengerTicketStatus[], string trainNO[], string
passengerArrivalCity[], string passengerDepartureCity[], string passengerTrainroute[], int
ticketPrice[], int idx, int delidx)
{
```

```
    for (int i = idx; i < delidx; i++)
    {
        passengerTicketStatus[i] = passengerTicketStatus[i + 1];
        trainNO[i] = trainNO[i + 1];
        passengerArrivalCity[i] = passengerArrivalCity[i + 1];
        passengerDepartureCity[i] = passengerDepartureCity[i + 1];
        passengerTrainroute[i] = passengerTrainroute[i + 1];
        ticketPrice[i] = ticketPrice[i + 1];
    }
}
void updateUserData(string userName[], string userID[], string userIDPassword[], string
userCnic[], string user2Cnic[], string uCode, int userCountIdx, int user2CountIdx)
{
    gotoxy(50, 23);
    cout << "Enter " << uCode << "ID:\e[0;32m ";
    string ID = inputs();

    if (userCheck(ID, userID, userCountIdx) == true)
    {
        int idx = indexCheck(ID, userID, userCountIdx);

        gotoxy(20, 23);
        cout << "\e[1;33mUser Information";
        viewUserData(userName, userID, userIDPassword, userCnic, idx, 15, 25);

        string option = updateDataChoice();
        string ind = updateData(userName, userIDPassword, userCnic, user2Cnic,
userCountIdx, user2CountIdx, idx, option);
```

```
        if (ind == "Y")
        {
            printStatement(uCode + " data updated successfully!", 50, 29, " ");
        }
        else
        {
            printStatement(uCode + " data not updated!", 50, 29, "W");
        }
    }
    else
    {
        printStatement(uCode + "ID not found!", 50, 25, "W");
    }
}
string updateDataChoice()
{
    string option;
    gotoxy(50, 25);
    cout << "\e[0;34mWhich attribute do you want to change?";
    gotoxy(50, 26);
    cout << "1.Name        2.CNIC       3.Exit";
    while (true)
    {
        space(63, 27);
        gotoxy(50, 27);
        cout << "\e[0;37mEnter option:\e[0;32m ";
        option = inputs();
        if (option == "1" || option == "2" || option =="3")
```

```
        {
            break;
        }
    }
    return option;
}
string updateData(string userName[], string userIDPassword[], string userCnic[], string user2Cnic[], int userCountIdx, int user2countIdx, int index, string option)
{
    string ind = " ";
    if (option == "1")
    {
        string newName = userNameSignUp(28);
        if (userNameValidationCheck(newName) == true)
        {
            userName[index] = newName;
            return "Y";
        }
    }
    else if (option == "2")
    {
        string newCnic = userCnicSignup(userCnic, user2Cnic, userCountIdx, user2countIdx, 28);
        if (userCnicValidationSignup(newCnic, userCnic, user2Cnic, userCountIdx, user2countIdx) == true)
        {
            userCnic[index] = newCnic;
            return "Y";
        }
```

```
    }
    else if (option=="3")
    {
        return " ";
    }
}
void viewUserDataList(string userName[], string userID[], string userIDPassword[], string userCnic[], string uCode, int userCountIdx)
{
    gotoxy(36, 23);
    cout << "\e[0;33m" << setfill(' ') << setw(20) << left << "Name" << setfill(' ') << setw(20) << left << "Cnic" << setfill(' ') << setw(20) << left << "UserID" << setfill(' ') << setw(20) << left << "Password\e[0;32m" << endl;
    if (userCountIdx > 0)
    {
        int y = 25;

        for (int idx = 0; idx < userCountIdx; idx++)
        {
            gotoxy(36, y);
            cout << setfill(' ') << setw(20) << left << userName[idx] << setfill(' ') << setw(20) << left << userCnic[idx] << setfill(' ') << setw(20) << left << userID[idx] << setfill(' ') << setw(20) << left << userIDPassword[idx] << endl;
            y = y + 1;
        }

        pressAnyKey(50, y + 1);
    }
    else
    {
```

```
            gotoxy(36, 25);
            cout << "\e[0;31mNo " << uCode << "ID exists.";
            gotoxy(36, 26);
            cout << "Add " << uCode << "s to view data!" << endl;

            pressAnyKey(50, 28);
        }
    }
    void viewUserData(string userName[], string userID[], string userIDPassword[], string
    userCnic[], int index, int x, int y)
    {
        gotoxy(x, y);
        cout << "\e[0;33mName:\e[0;32m " << userName[index];
        gotoxy(x, y + 1);
        cout << "\e[0;33mUserID:\e[0;32m " << userID[index];
        gotoxy(x, y + 2);
        cout << "\e[0;33mPassword:\e[0;32m " << userIDPassword[index];
        gotoxy(x, y + 3);
        cout << "\e[0;33mCnic:\e[0;32m " << userCnic[index];
    }
    void searchUserData(string userName[], string userID[], string userIDPassword[], string
    userCnic[], string uCode, int userCountIdx)
    {
        gotoxy(50, 23);
        cout << "Enter " << uCode << "ID:\e[0;32m ";
        string ID = inputs();

        if (userCheck(ID, userID, userCountIdx) == true)
        {
```

```
            int idx = indexCheck(ID, userID, userCountIdx);

            viewUserData(userName, userID, userIDPassword, userCnic, idx, 50, 25);
            pressAnyKey(50, 30);
        }
        else
        {
            printStatement(uCode + " data not found!", 50, 25, "W");
        }
    }
    string changePassword(string idPassword[], int index)
    {
        string password;
        gotoxy(50, 23);
        cout << "Enter current password:\e[0;32m ";
        password = inputs();
        string newPassword;
        if (password == idPassword[index])
        {
            newPassword = userPasswordSignup(25);
            bool validation = passwordValidationCheckSignup(newPassword);

            if (validation == true && newPassword != password)
            {
                idPassword[index] = newPassword;
                printStatement("Password changed successfully!", 50, 26, " ");
                return newPassword;
            }
```

```
        else if (newPassword == password)
        {
            printStatement("Existing password cannot be used! Password not changed!", 50, 26, "W");
            return " ";
        }
        else
        {
            printStatement("Invalid Password format! Password not changed!", 50, 26, "W");
            return " ";
        }
    }
    else
    {
        printStatement("Wrong credentials! Password not changed!", 50, 24, "W");
        return " ";
    }
}


// train addition removal functions
void addTrainRoute(string trainNo[], string trainArrivalCity[], string trainDepartureCity[], string trainRoute[], int ticketPrice[], int &trainCountIdx, string cities[])
{
    string trainID = trainNoInput(trainNo, trainCountIdx);
    bool validation = trainNoValidation(trainID, trainNo, trainCountIdx);

    if (validation)
    {
```

```
        trainDepartureCity[trainCountIdx] = trainDepartureCityInput(cities);
        trainArrivalCity[trainCountIdx] =
trainArrivalCityInput(trainDepartureCity[trainCountIdx], cities);
        trainRoute[trainCountIdx] = trainDepartureCity[trainCountIdx] + "-to-" +
trainArrivalCity[trainCountIdx];
        trainNo[trainCountIdx] = trainID;


        gotoxy(50, 26);
        cout << "\e[0;37mRoute of train " << trainNo[trainCountIdx] << ":\e[0;32m " <<
trainRoute[trainCountIdx];
        ticketPrice[trainCountIdx] = trainTicketPriceIn();
        trainCountIdx++;


        printStatement("Train data added successfully!", 50, 28, " ");
    }
    else
    {
        printStatement("Invalid trainNO! Train route not added!", 50, 24, "W");
    }
}
string trainNoInput(string trainNo[], int trainCountIdx)
{
    noteAddTrain();
    string trainNoIn;


    for (int i = 0; i < 3; i++)
    {
        if (i == 0)
        {
```

```
            space(64, 23);
            gotoxy(50, 23);
            cout << "\e[0;37mEnter trainNo:\e[0;32m ";
            trainNoIn = inputs();
        }
        else
        {
            space(80, 23);
            gotoxy(50, 23);
            cout << "\e[0;31mInvalid trainNo! \e[0;37mEnter again:\e[0;32m ";
            trainNoIn = inputs();
        }
        if (trainNoValidation(trainNoIn, trainNo, trainCountIdx) == true)
        {
            break;
        }
    }

    eraseInstruction();

    return trainNoIn;
}
bool trainNoValidation(string trainNoIn, string trainNo[], int trainCountIdx)
{
    bool check = true;

    if (trainNoIn.length() >= 4 && trainNoIn.length() <= 12)
    {
```

```
        int i = 0;
        while (trainNoIn[i] != '\0')
        {
            if (!((trainNoIn[i] >= 'a' && trainNoIn[i] <= 'z') || (trainNoIn[i] >= 'A' &&
trainNoIn[i] <= 'Z') || (trainNoIn[i] >= '0' && trainNoIn[i] <= '9') || trainNoIn[i] == '-'))
            {
                check = false;
                break;
            }
            i++;
        }
        if (check == true)
        {
            check = trainCheck(trainNoIn, trainNo, trainCountIdx);
        }
    }
    else
    {
        check = false;
    }

    return check;
}
string trainDepartureCityInput(string cities[])
{
    noteDepartureCity(cities);

    string departureCity;
```

```
    while (true)
    {
        space(72, 24);
        gotoxy(50, 24);
        cout << "\e[0;37mEnter Departure City: \e[0;32m";
        departureCity = inputs();
        if (cityNameValidation(departureCity, cities) == true)
        {
            break;
        }
    }

    return departureCity;
}
bool trainCheck(string trainNoIn, string trainNo[], int trainCountIdx)
{
    bool check = true;
    if (trainCountIdx > 0)
    {
        for (int idx = 0; idx < trainCountIdx; idx++)
        {
            if (trainNoIn == trainNo[idx])
            {
                check = false;
                break;
            }
        }
    }
```

```
    else
    {
        check = true;
    }
    return check;
}
string trainArrivalCityInput(string departureCity, string cities[])
{
    noteArrivalCity(departureCity, cities);
    string arrivalCity;

    while (true)
    {
        space(70, 25);
        gotoxy(50, 25);
        cout << "\e[0;37mEnter Arrival City: \e[0;32m";
        arrivalCity = inputs();
        if (cityNameValidation(arrivalCity, cities) == true && arrivalCity != departureCity)
        {
            break;
        }
    }

    return arrivalCity;
}
bool cityNameValidation(string city, string cities[])
{
    bool check = false;
```

```
    for (int i = 0; i < 15; i++)
    {
        if (city == cities[i])
        {
            check = true;
            break;
        }
    }
    return check;
}
int trainTicketPriceIn()
{
    string price;
    while (true)
    {
        space(70, 27);
        gotoxy(50, 27);
        cout << "\e[0;37mEnter ticket price:\e[0;32m ";
        price = inputs();

        if (ticketPriceValidation(price) == true)
        {
            break;
        }
    }

    int trainTicketprice = stringToIntConversion(price);
```

```
    return trainTicketprice;
}
bool ticketPriceValidation(string price)
{
    bool check = true;

    int i = 0;
    while (price[i] != '\0')
    {
        if (price[i] < '0' || price[i] > '9')
        {
            check = false;
            break;
        }
        i++;
    }

    return check;
}
int stringToIntConversion(string price)
{
    double tPrice = 0;
    int i = 0;
    int j = price.length() - 1;
    while (i < price.length())
    {
        tPrice = tPrice + (price[i] - '0') * pow(10, j);
```

```
        i++;
        j = j - 1;
    }

    int ticketPrice = ceil(tPrice);
    return ticketPrice;
}
void deleteTrainTicketDetails(string passengerTicketStatus[], string passengerTrainNo[], string passengerArrivalCity[], string passengerDepartureCity[], string passengerTicketRoute[], int passengerticketPrice[], int passengerCountIdx, string trainID)
{
    for (int idx = 0; idx < passengerCountIdx; idx++)
    {
        if (passengerTrainNo[idx] == trainID)
        {
            passengerTicketRoute[idx] = "N.A";
            passengerDepartureCity[idx] = "N.A";
            passengerArrivalCity[idx] = "N.A";
            passengerTrainNo[idx] = "N.A";
            passengerTicketStatus[idx] = "N.A";
            passengerticketPrice[idx] = 0.0;
        }
    }
}
string deleteTrainRoute(string trainNo[], string trainArrivalCity[], string trainDepartureCity[], string trainRoute[], int ticketPrice[], int &trainCountIdx)
{
    gotoxy(50, 23);
    cout << "\e[0;37mEnter trainNo: \e[0;32m";
```

```
    string trainCode = inputs();


    if (trainCountIdx > 0)
    {
        if (trainCheck(trainCode, trainNo, trainCountIdx) == false)
        {
            int idx = indexCheck(trainCode, trainNo, trainCountIdx);
            viewTicketDetails(trainNo, trainRoute, trainArrivalCity, trainDepartureCity,
ticketPrice, idx);
            gotoxy(50, 28);
            cout << "\e[0;34mDo you want to delete train data?";
            gotoxy(50, 29);
            cout << "     1.Yes       2.No";
            string option;
            while (true)
            {
                space(64, 30);
                gotoxy(50, 30);
                cout << "\e[0;37mEnter option:\e[0;32m ";
                option = inputs();
                if (option == "Yes" || option == "No" || option == "1" || option == "2")
                {
                    break;
                }
            }
            if (option == "1" || option == "Yes")
            {
                deleteData(trainNo, trainArrivalCity, trainDepartureCity, trainRoute,
ticketPrice, idx, trainCountIdx);
```

```
            printStatement("Train data deleted successfully!", 50, 31, " ");
        }
        else
        {
            trainCode = " ";
            printStatement("Train data not deleted!", 50, 31, " ");
        }
    }
    else
    {
        trainCode = " ";
        printStatement("Train data not found!", 50, 25, "W");
    }
}
else
{
    trainCode = " ";
    printStatement("Train data not found!", 50, 25, "W");
}

return trainCode;
}
void deleteData(string trainNo[], string trainArrivalCity[], string trainDepartureCity[],
string trainRoute[], int ticketPrice[], int index, int &trainCountIdx)
{
for (int i = index; i < trainCountIdx; i++)
{
    trainNo[i] = trainNo[i + 1];
    trainArrivalCity[i] = trainArrivalCity[i + 1];
```

```
        trainDepartureCity[i] = trainDepartureCity[i + 1];
        trainRoute[i] = trainRoute[i + 1];
        ticketPrice[i] = ticketPrice[i + 1];
    }
    trainCountIdx = trainCountIdx - 1;
}
void viewTrainsAvailable(string trainNo[], string trainArrivalCity[], string
trainDepartureCity[], string trainRoute[], int trainTicketPrice[], int trainCountIdx)
{
    gotoxy(30, 23);
    cout << "\e[0;33m" << setfill(' ') << setw(20) << left << "TrainNo" << setfill(' ') <<
setw(20) << left << "DepartureCity" << setfill(' ') << setw(20) << left << "Arrival City"
<< setfill(' ') << setw(30) << left << "Route"
         << "TicketPrice\e[0;32m" << endl;
    if (trainCountIdx > 0)
    {
        int y = 25;

        for (int idx = 0; idx < trainCountIdx; idx++)
        {
            gotoxy(30, y);
            cout << setfill(' ') << setw(20) << left << trainNo[idx] << setfill(' ') << setw(20)
<< left << trainDepartureCity[idx] << setfill(' ') << setw(20) << left <<
trainArrivalCity[idx] << setfill(' ') << setw(30) << left << trainRoute[idx] <<
trainTicketPrice[idx] << endl;
            y = y + 1;
        }

        pressAnyKey(50, y + 1);
    }
```

```
    else
    {
        gotoxy(50, 25);
        cout << "\e[0;31mNo train data available.";
        pressAnyKey(50, 26);
    }
}


// ticket related functions
void bookTickets(string passengerTicketStatus[], string passengerTrainNo[], string
trainNo[], string passengerArrivalCity[], string trainArrivalCity[], string
passengerDepartureCity[], string trainDepartureCity[], string passengerTicketRoute[],
string trainRoute[], int passengerTicketPrice[], int trainTicketPrice[], int index, int
tCount)
{
    if (passengerTicketStatus[index] == "Y")
    {
        printStatement("Ticket is already booked! Cancel ticket to book new one!", 50, 24,
"W");
    }
    else
    {
        noteRoutesavail(trainNo, trainRoute, tCount);
        string routeNo;
        int indexTrain;
        while (true)
        {
            space(96, 24);
            gotoxy(50, 24);
            cout << "\e[0;37mEnter the Sr.No. for the train to book ticket: \e[0;32m";
```

```
            routeNo = inputs();
            if (ticketPriceValidation(routeNo) == true && stringToIntConversion(routeNo)
<= tCount && stringToIntConversion(routeNo) > 0)
            {
               break;
            }
            else if (routeNo == "esc")
            {
               break;
            }
         }
         if (routeNo == "esc")
         {
            printStatement("Ticket not booked!", 50, 25, "W");
         }
         else
         {
            indexTrain = stringToIntConversion(routeNo) - 1;
            saveTicketData(passengerTicketStatus, passengerTrainNo, trainNo,
passengerArrivalCity, trainArrivalCity, passengerDepartureCity, trainDepartureCity,
passengerTicketRoute, trainRoute, passengerTicketPrice, trainTicketPrice, index,
indexTrain);


            printStatement("Ticket booked successfully!", 50, 25, " ");
         }
      }
   }
   void saveTicketData(string passengerTicketStatus[], string passengerTrainNo[], string
trainNo[], string passengerArrivalCity[], string trainArrivalCity[], string
passengerDepartureCity[], string trainDepartureCity[], string passengerTicketRoute[],
```

```
string trainRoute[], int passengerTicketPrice[], int trainTicketPrice[], int index, int
indexTrain)
{
   passengerTicketStatus[index] = "Y";
   passengerTrainNo[index] = trainNo[indexTrain];
   passengerTicketRoute[index] = trainRoute[indexTrain];
   passengerArrivalCity[index] = trainArrivalCity[indexTrain];
   passengerDepartureCity[index] = trainDepartureCity[indexTrain];
   passengerTicketPrice[index] = trainTicketPrice[indexTrain];
}
void cancelTicket(string passengerTicketStatus[], string passengerTrainNo[], string
passengerTicketRoute[], string passengerArrivalCity[], string passengerDepartureCity[],
int passengerTicketPrice[], int index)
{
   viewTicketDetails(passengerTrainNo, passengerTicketRoute, passengerArrivalCity,
passengerDepartureCity, passengerTicketPrice, index);

   gotoxy(50, 28);
   cout << "\e[0;34mAre you sure you want to cancel ticket?";
   gotoxy(50, 29);
   cout << "      1.Yes          2.No";
   string option;
   while (true)
   {
      space(64, 30);
      gotoxy(50, 30);
      cout << "\e[0;37mEnter option:\e[0;32m ";
      option = inputs();
      if (option == "Yes" || option == "No" || option == "1" || option == "2")
      {
```

```
            break;
        }
    }
    if (option == "Yes" || option == "1")
    {
        ticketStatusPassenger(passengerTicketStatus, passengerTrainNo,
passengerTicketRoute, passengerArrivalCity, passengerDepartureCity,
passengerTicketPrice, index);
        printStatement("Ticket cancelled successfully!", 50, 31, " ");
    }
    else
    {
        printStatement("Ticket not cancelled!", 50, 31, " ");
    }
}
void viewTicketDetails(string trainNo[], string ticketRoute[], string arrivalCity[], string
departureCity[], int ticketPrice[], int index)
{
    gotoxy(25, 25);
    cout << "\e[0;33m" << setfill(' ') << setw(20) << left << "TrainNo" << setfill(' ') <<
setw(30) << left << "Ticket Route" << setfill(' ') << setw(20) << left << "ArrivalCity" <<
setfill(' ') << setw(20) << left << "DepartureCity" << setfill(' ') << setw(20) << left <<
"TicketPrice" << endl;
    gotoxy(25, 26);
    cout << "\e[0;32m" << setfill(' ') << setw(20) << left << trainNo[index] << setfill(' ')
<< setw(30) << left << ticketRoute[index] << setfill(' ') << setw(20) << left <<
arrivalCity[index] << setfill(' ') << setw(20) << left << departureCity[index] << setfill(' ')
<< setw(20) << left << ticketPrice[index] << endl;
}
void viewBookedTickets(string passengerName[], string passengerCnic[], string
passengerTicketStatus[], string passengerTrainNo[], string passengerTicketRoute[], string
```

```
passengerArrivalCity[], string passengerDepartureCity[], int passengerTicketPrice[], int index)
{
   if (index > 0)
   {
      gotoxy(12, 23);
      cout << "\e[0;33m" << setfill(' ') << setw(20) << left << "Name" << setfill(' ') << setw(20) << left << "Passenger Cnic" << setfill(' ') << setw(30) << left << "TicketRoute" << setfill(' ') << setw(20) << left << "TrainNo" << setfill(' ') << setw(20) << left << "DepartureCity" << setfill(' ') << setw(20) << left << "ArrivalCity" << setfill(' ') << setw(20) << left << "TicketPrice" << endl;
      int y = 25;
      int j = 0;
      for (int i = 0; i < index; i++)
      {
         if (passengerTicketStatus[i] == "Y")
         {
            gotoxy(12, y);
            cout << "\e[0;32m" << setfill(' ') << setw(20) << left << passengerName[i] << setfill(' ') << setw(20) << left << passengerCnic[i] << setfill(' ') << setw(30) << left << passengerTicketRoute[i] << setfill(' ') << setw(20) << left << passengerTrainNo[i] << setfill(' ') << setw(20) << left << passengerDepartureCity[i] << setfill(' ') << setw(20) << left << passengerArrivalCity[i] << setfill(' ') << setw(20) << left << passengerTicketPrice[i] << endl;
            y = y + 1;
            j++;
         }
      }
      if (j == 0)
      {
         printStatement("No tickets data available!", 50, 25, "W");
      }
```

```
        else
        {
            pressAnyKey(50, y + 1);
        }
    }
    else
    {
        printStatement("No tickets data available!", 50, 25, "W");
    }
}
void ticketsDetails(string passengerTrainNo[], string trainNo[], int
passengerTicketPrice[], int trainTicketPrice[], int passengerCountIdx, int trainCountIdx)
{
    int totalRevenue = 0;
    int totalTicketsSold = 0;
    int numOfBookedTickets[trainCountIdx];
    int revenueOfEachTrain[trainCountIdx];
    if (trainCountIdx > 0 && passengerCountIdx > 0)
    {
        for (int i = 0; i < trainCountIdx; i++)
        {
            numOfBookedTickets[i] = 0;
            revenueOfEachTrain[i] = 0;

            for (int j = 0; j < passengerCountIdx; j++)
            {
                if (passengerTrainNo[j] == trainNo[i])
                {
                    numOfBookedTickets[i]++;
```

```
                totalRevenue = totalRevenue + passengerTicketPrice[j];
                totalTicketsSold = totalTicketsSold + 1;
            }
        }
        revenueOfEachTrain[i] = trainTicketPrice[i] * numOfBookedTickets[i];
    }
    int y = 25;
    gotoxy(45, 24);
    cout << "\e[0;33m" << setfill(' ') << setw(20) << left << "TrainNo" << setfill(' ') << setw(25) << left << "No. of tickets booked" << setfill(' ') << setw(20) << left << "Revenue Collected";
    for (int i = 0; i < trainCountIdx; i++)
    {
        gotoxy(45, y);
        cout << "\e[0;32m" << setfill(' ') << setw(20) << left << trainNo[i] << setfill(' ') << setw(25) << left << numOfBookedTickets[i] << "Rs." << revenueOfEachTrain[i];
        y++;
    }
    gotoxy(50, y + 1);
    cout << "\e[0;35mTotal Booked Tickets: " << totalTicketsSold;
    gotoxy(50, y + 2);
    cout << "\e[0;35mTotal Revenue Collected: " << totalRevenue;
    pressAnyKey(50, y + 3);
  }
  else
  {
    printStatement("Data not available", 50, 24, "W");
  }
}
```

```
// employee file handeling functions

void employeesNewDataFile(string employeeName[], string employeeId[], string
employeeIDPassword[], string employeeCnic[], int employeeCountIdx)

{

   fstream employeeFile;

   employeeFile.open("employeeData.txt", ios::app);


   employeeFile << employeeName[employeeCountIdx - 1] + ';' +
employeeId[employeeCountIdx - 1] + ';' + employeeIDPassword[employeeCountIdx - 1]
+ ';' + employeeCnic[employeeCountIdx - 1] + ";;" << endl;

   employeeFile.close();

}

void employeesDataUpdateFile(string employeeName[], string employeeId[], string
employeeIDPassword[], string employeeCnic[], int employeeCountIdx)

{

   fstream employeeFile;

   employeeFile.open("employeeData.txt", ios::out);


   for (int i = 0; i < employeeCountIdx; i++)

   {

      employeeFile << employeeName[i] + ';' + employeeId[i] + ';' +
employeeIDPassword[i] + ';' + employeeCnic[i] + ";;" << endl;

   }


   employeeFile.close();

}

void employeeDataLoad(string employeeName[], string employeeId[], string
employeeIDPassword[], string employeeCnic[], int &employeeCountIdx)

{
```

```
    string line;
    fstream employeeFile;
    employeeFile.open("employeeData.txt", ios::in);
    int i = 0;
    while (getline(employeeFile, line))
    {
        int j = 0;
        if (line[line.length() - 1] == ';' && line[line.length() - 2] == ';' && line[line.length() - 16] == ';' && line[0] != ';')
        {
            employeeName[i] = loadUserAttribute(line, j);
            j++;
            employeeId[i] = loadUserAttribute(line, j);
            j++;
            employeeIDPassword[i] = loadUserAttribute(line, j);
            j++;
            employeeCnic[i] = loadUserAttribute(line, j);
            i++;
        }
    }
    employeeCountIdx = i;
    employeeFile.close();
}


// passenger file handeling functions
void passengersNewDataFile(string passengerName[], string passengerID[], string passengerIDPassword[], string passengerCnic[], string passengerTicketStatus[], string passengerTrainNo[], string passengerTicketRoute[], string passengerArrivalCity[], string passengerDepartureCity[], int passengerTicketPrice[], int passengerCountIdx)
{
```

```
    fstream passengerFile;
    passengerFile.open("passengerData.txt", ios::app);

    passengerFile << passengerName[passengerCountIdx - 1] + ';' +
passengerID[passengerCountIdx - 1] + ';' + passengerIDPassword[passengerCountIdx -
1] + ';' + passengerCnic[passengerCountIdx - 1] + ';' +
passengerTicketStatus[passengerCountIdx - 1] + ';' +
passengerTrainNo[passengerCountIdx - 1] + ';' +
passengerTicketRoute[passengerCountIdx - 1] + ';' +
passengerArrivalCity[passengerCountIdx - 1] + ';' +
passengerDepartureCity[passengerCountIdx - 1] + ';' +
to_string(passengerTicketPrice[passengerCountIdx - 1]) + ";;" << endl;
    passengerFile.close();
}
void passengersDataUpdateFile(string passengerName[], string passengerID[], string
passengerIDPassword[], string passengerCnic[], string passengerTicketStatus[], string
passengerTrainNo[], string passengerTicketRoute[], string passengerArrivalCity[], string
passengerDepartureCity[], int passengerTicketPrice[], int passengerCountIdx)
{
    fstream passengerFile;
    passengerFile.open("passengerData.txt", ios::out);

    for (int i = 0; i < passengerCountIdx; i++)
    {
        passengerFile << passengerName[i] + ';' + passengerID[i] + ';' +
passengerIDPassword[i] + ';' + passengerCnic[i] + ';' + passengerTicketStatus[i] + ';' +
passengerTrainNo[i] + ';' + passengerTicketRoute[i] + ';' + passengerArrivalCity[i] + ';' +
passengerDepartureCity[i] + ';' + to_string(passengerTicketPrice[i]) + ";;" << endl;
    }

    passengerFile.close();
}
```

```
void passengersDataLoad(string passengerName[], string passengerID[], string passengerIDPassword[], string passengerCnic[], string passengerTicketStatus[], string passengerTrainNo[], string passengerTicketRoute[], string passengerArrivalCity[], string passengerDepartureCity[], int passengerTicketPrice[], int &passengerCountIdx)
{
    string line;
    fstream passengerFile;
    passengerFile.open("passengerData.txt", ios::in);
    int i = 0;
    while (getline(passengerFile, line))
    {
        int j = 0;
        if (line[line.length() - 1] == ';' && line[line.length() - 2] == ';' && line[0] != ';')
        {
            passengerName[i] = loadUserAttribute(line, j);
            j++;
            passengerID[i] = loadUserAttribute(line, j);
            j++;
            passengerIDPassword[i] = loadUserAttribute(line, j);
            j++;
            passengerCnic[i] = loadUserAttribute(line, j);
            j++;
            passengerTicketStatus[i] = loadUserAttribute(line, j);
            j++;
            passengerTrainNo[i] = loadUserAttribute(line, j);
            j++;
            passengerTicketRoute[i] = loadUserAttribute(line, j);
            j++;
            passengerArrivalCity[i] = loadUserAttribute(line, j);
```

```
            j++;
            passengerDepartureCity[i] = loadUserAttribute(line, j);
            j++;
            passengerTicketPrice[i] = stringToIntConversion(loadUserAttribute(line, j));
            i++;
        }
    }
    passengerCountIdx = i;
    passengerFile.close();
}

// train file handeling functions
void trainsNewDataFile(string trainNo[], string trainRoute[], string trainArrivalCity[],
string trainDepartureCity[], int ticketPrice[], int trainCountIdx)
{
    fstream trainFile;
    trainFile.open("trainData.txt", ios::app);

    trainFile << trainNo[trainCountIdx - 1] + ';' + trainRoute[trainCountIdx - 1] + ';' +
trainArrivalCity[trainCountIdx - 1] + ';' + trainDepartureCity[trainCountIdx - 1] + ';' +
to_string(ticketPrice[trainCountIdx - 1]) + ";;" << endl;
    trainFile.close();
}
void trainsDataUpdateFile(string trainNo[], string trainRoute[], string trainArrivalCity[],
string trainDepartureCity[], int ticketPrice[], int trainCountIdx)
{
    fstream trainFile;
    trainFile.open("trainData.txt", ios::out);

    for (int i = 0; i < trainCountIdx; i++) // stores all the
```

```
    {
        trainFile << trainNo[i] + ';' + trainRoute[i] + ';' + trainArrivalCity[i] + ';' +
trainDepartureCity[i] + ';' + to_string(ticketPrice[i]) + ";;" << endl;
    }

    trainFile.close();
}
void trainsDataLoad(string trainNo[], string trainRoute[], string trainArrivalCity[], string
trainDepartureCity[], int ticketPrice[], int &trainCountIdx)
{
    string line;
    fstream trainFile;
    trainFile.open("trainData.txt", ios::in);
    int i = 0;
    while (getline(trainFile, line)) // loop terminates if file ends
    {
        int j = 0;
        if (line[line.length() - 1] == ';' && line[line.length() - 2] == ';' && line[0] != ';')
        {
            trainNo[i] = loadUserAttribute(line, j);
            j++;
            trainRoute[i] = loadUserAttribute(line, j);
            j++;
            trainArrivalCity[i] = loadUserAttribute(line, j);
            j++;
            trainDepartureCity[i] = loadUserAttribute(line, j);
            j++;
            ticketPrice[i] = stringToIntConversion(loadUserAttribute(line, j));
            i++;
```

```
        }
    }
    trainCountIdx = i;
    trainFile.close();
}

// loads each attribute of a user
string loadUserAttribute(string line, int &idx)
{
    string n = "";
    while (line[idx] != ';') // stores the word character by character till semicolon is found
    {
        n = n + line[idx];
        idx++;
    }
    return n;
}

// redudant functions
void gotoxy(int x, int y)
{
    COORD coordinates;
    coordinates.X = x;
    coordinates.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinates);
}
void pressAnyKey(int x, int y)
{
```

```
    gotoxy(x, y);
    cout << "\e[0;37mPress any key to continue...";
    getch();
}
void space(int x, int y)
{
    gotoxy(x, y);
    cout << "                                        ";
}
string inputs()
{
    string input;
    while (!(input.length() >= 1))
    {
        getline(cin, input);
    }
    return input;
}
string userIDInput(int i, int y)
{
    string userName;

    if (i == 0) // if user enters wrong userID
    {
        gotoxy(50, y);
        cout << "\e[0;37mEnter userID:\e[0;32m ";
        userName = inputs();
    }
```

```
    else // if user enters wrong userId
    {
        space(76, y);
        gotoxy(50, y);
        cout << "\e[0;31mInvalid userID! \e[0;37mEnter again:\e[0;32m ";
        userName = inputs();
    }

    return userName;
}
void headerCls()
{
    system("cls");
    header();
}
void printStatement(string statement, int x, int y, string ind)
{
    if (ind == "W")
    {
        cout << "\e[0;31m";
    }
    else
    {
        cout << "\e[0;32m";
    }
    gotoxy(x, y);
    cout << statement;
    pressAnyKey(50, y + 1);
```

}

9. Weakness in the Business Application

The weaknesses in the business application are:

- A passenger can book a ticket for himself only.
- A user cannot book more than 1 tickets.
- Train data cannot be updated but deleted.
- Tickets details cannot be updated but deleted.
- There is no defined ticket type for train.
- There is no seats limit for the trains.
- A user cannot contact admin or employee to submit complaint.
- If password is forgotten you cannot recover it.

10. Future Directions

I want to extend this application and add some more features i.e., booking tickets for other users, password recovery, limiting trains seats, message box etc. so that it can be more efficient in its working.