**Train Management System**



Session: 2022 – 2025

**Submitted by:**

Muhammad Tabish Akhtar 2022-CS-78

**Supervised by:**

Sir Irzam and Ma’m Maida Shaid

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

* **Short Description of your project**
  + I am trying to make a business application which can help a user to buy tickets , caculate fair of Journary , view available trains detail , and Train timings. When a manager of Trains login to portal , he can set train timings ,add drivers detail, update driver detail , delete drivers detail, can set returnable amount if a user cancel his ticket. It would be Helpful in computer science field because after this project we can manage our train tickets , pay bill online or cash instantly at any place online by that application.
* **Users of Application (minimum 2 users for your project)**
  + Admin:The main Controller of Application is Its Admin who want to control his trains,and to run his business.
  + Employees: The people who have to travel through that trains are Employees.They can get Tickets by paying ticket amount.
* **Functional Requirements**
  + Functional requirements are a detailed description of what a system, product, or service must do to meet the needs and expectations
  + Use the following format to write the Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| User Story ID | *As a* | *I want to perform* | *So that I can* |
| **1** | Admin | Add Trains | Enter Train names and routes to add them. |
|  |  | Set Fair | Set Fare of registered Trains by entering their names. |
|  |  | Set Time | Set Timing of Registed Trains By entering their names. |
|  |  | Update time | Update Train Timing by entering train number. |
|  |  | Remove Train | Remove Train By entering train Name. |
|  |  | Add Driver | Add Driver to a train by entering its name. |
|  |  | Update driver details | Update Driver of a train by entering its train number. |
|  |  | Remove Driver Details | Remove Driver Details by entering train number. |
|  |  | View all trains Details | View all Registered trains,Their Fares and driver details. |
|  |  | Set Returnable Rules | Set Rules for Cancelation of Ticket. |
|  |  |  |  |

* + Use the following format to write the Functional Requirements

|  |  |  |  |
| --- | --- | --- | --- |
| User Story ID | *As a* | *I want to perform* | *So that I can* |
| **2** | User | See All available Trains | See all available trains name |
|  |  | See all Train Fares | See all Train Fares with names |
|  |  | See all Train Timings | See all Train Timings with names. |
|  |  | Book Ticket | Enter train number and class type and number of tickets then I book my ticket |
|  |  | Generate Invoice Bill | Enter Train Number if I book Its ticket then it prints Invoice Bill on Screen |
|  |  | Cancel Ticket | Enter train number and cancel my ticket. |

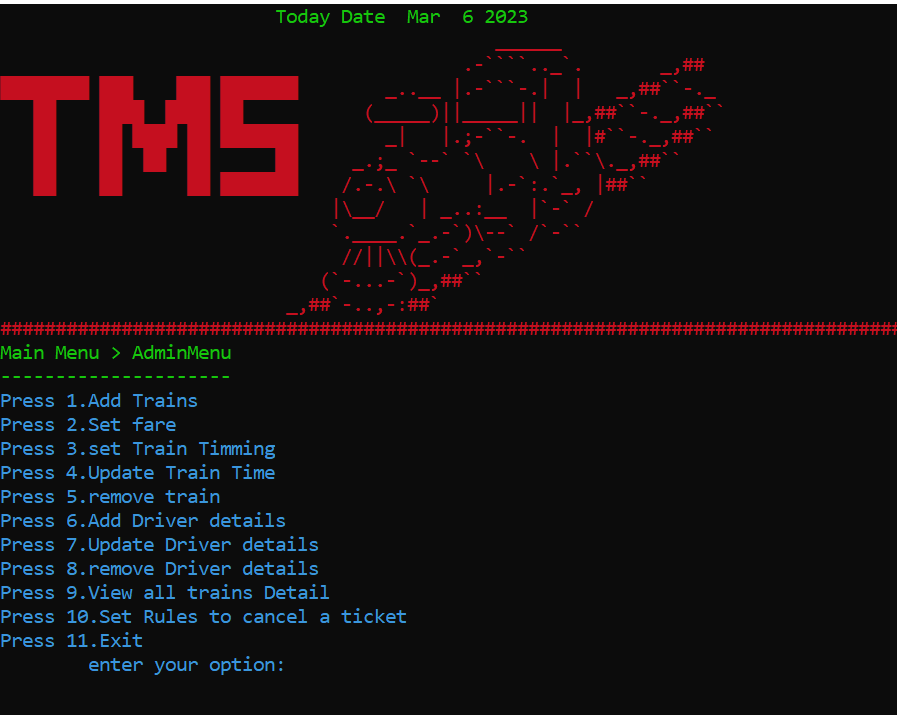
**Wireframes**

****

**Figure 1: Welcome Screen**

****

**Figure 2: Login Screen**

****

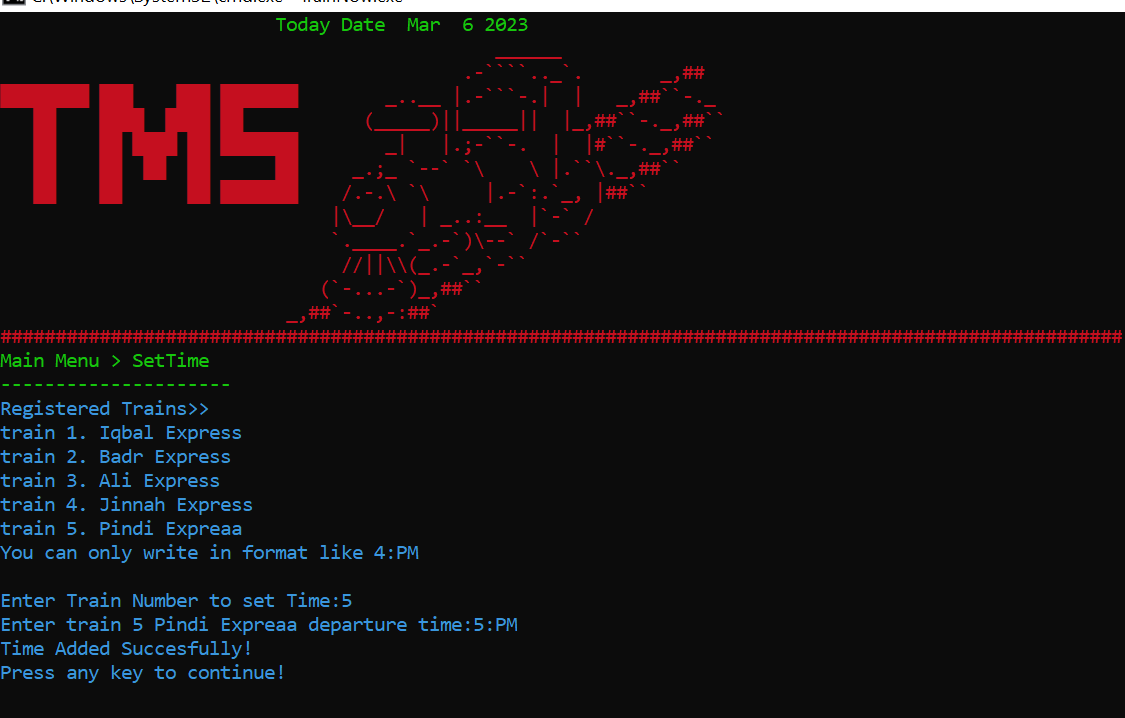
**Figure 3: Admin Main Menu Screen**

****

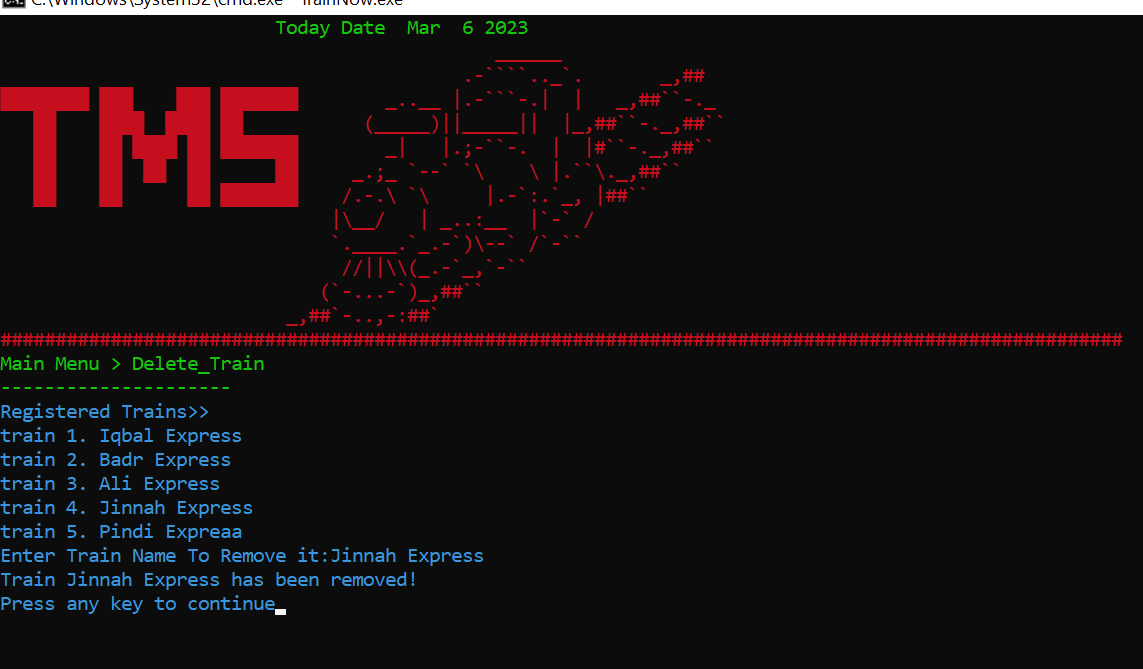
**Figure 4: Add Train by Admin Screen**

****

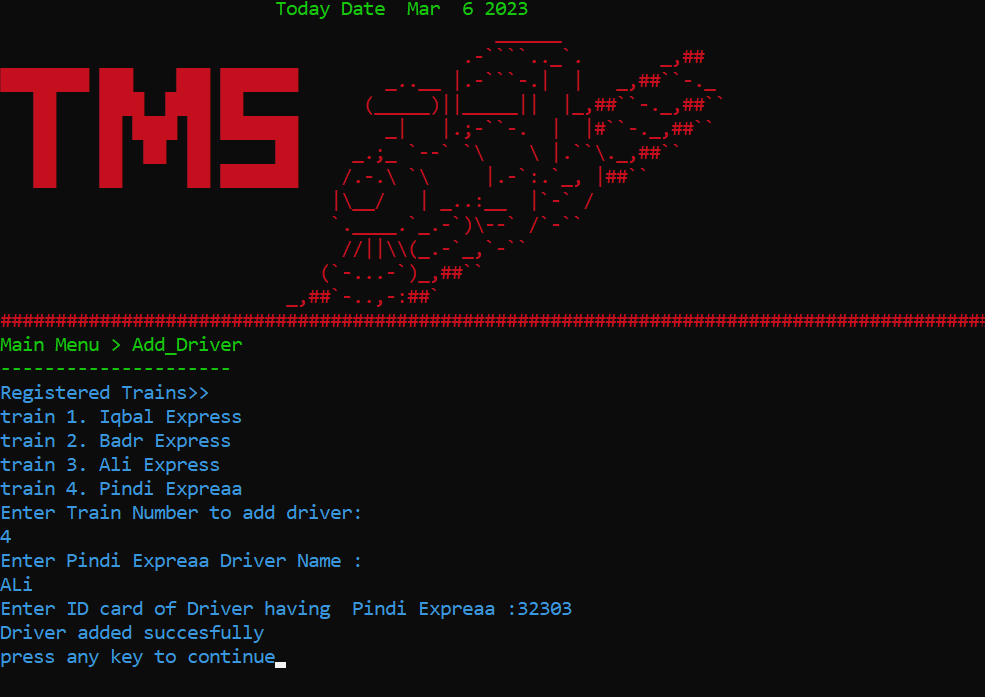
**Figure 5: Fare set by Admin**

****

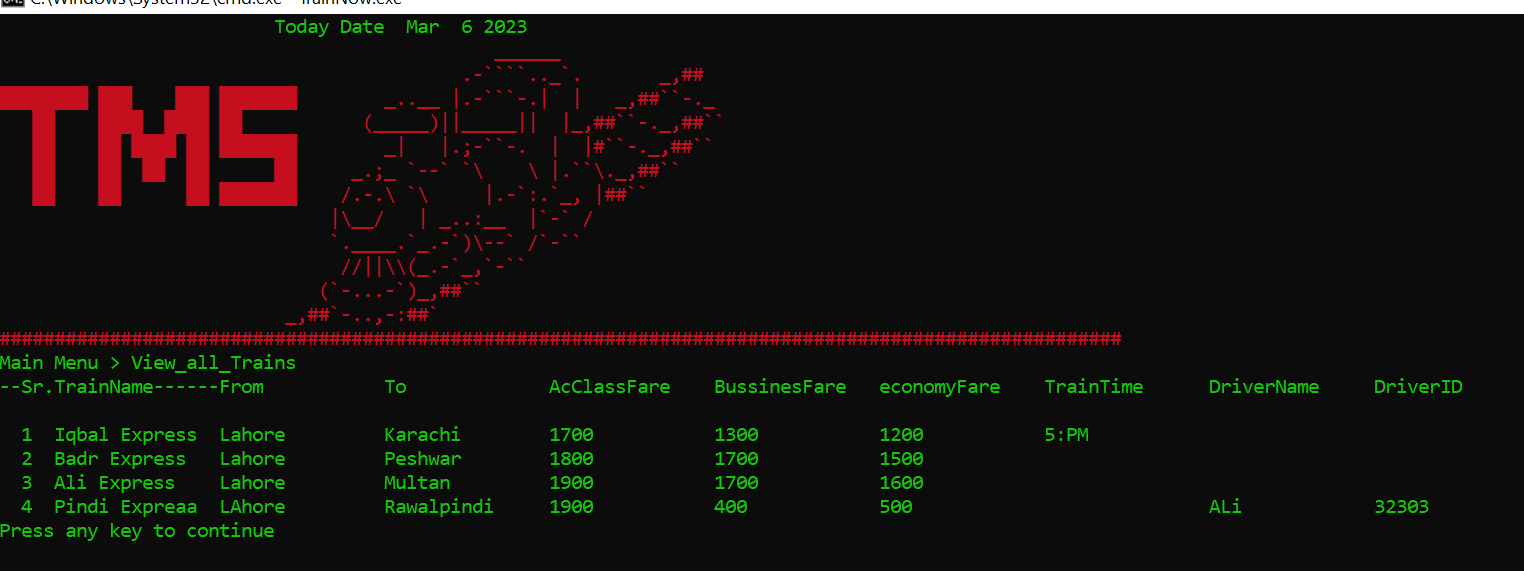
**Figure 5: Time set by Admin**

****

**Figure 5: Train Deleted by Admin**

****

**Figure 5: Driver Details added by Admin**

****

**Figure 5: View all Details by Admin**

****

**Figure 5: Main Menu of User**

****

**Figure 5: All Trains Timming seen of User**

****

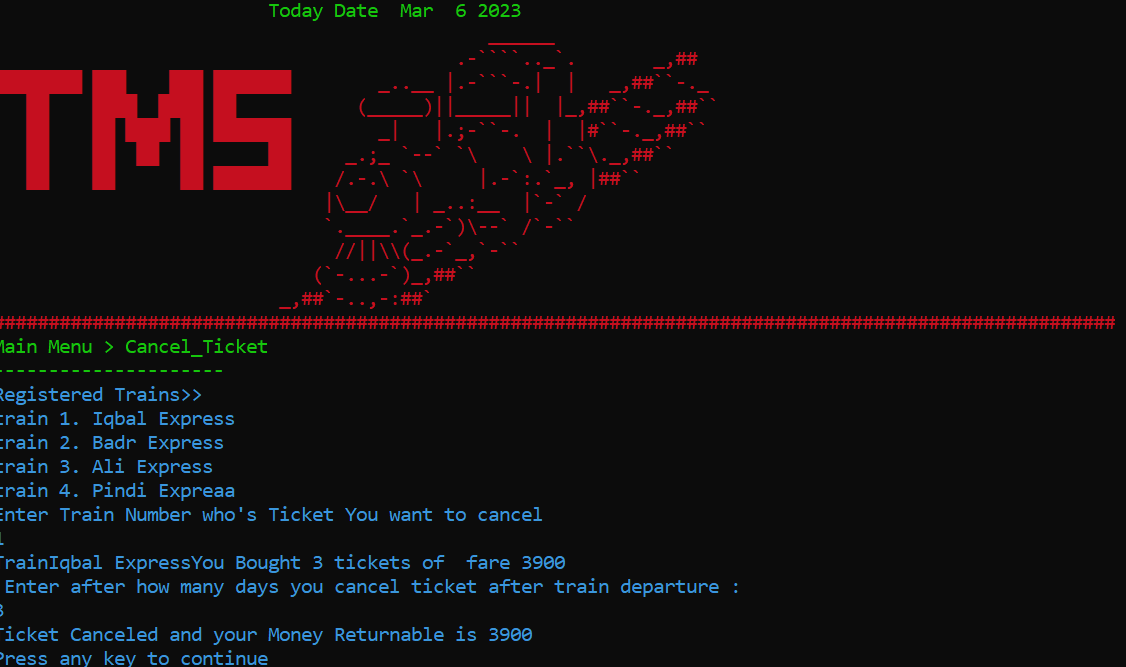
**Figure 5: All Trains Name seen of User**

****

**Figure 5: Ticket Book by User**

****

**Figure 5: Bill Generated by User**

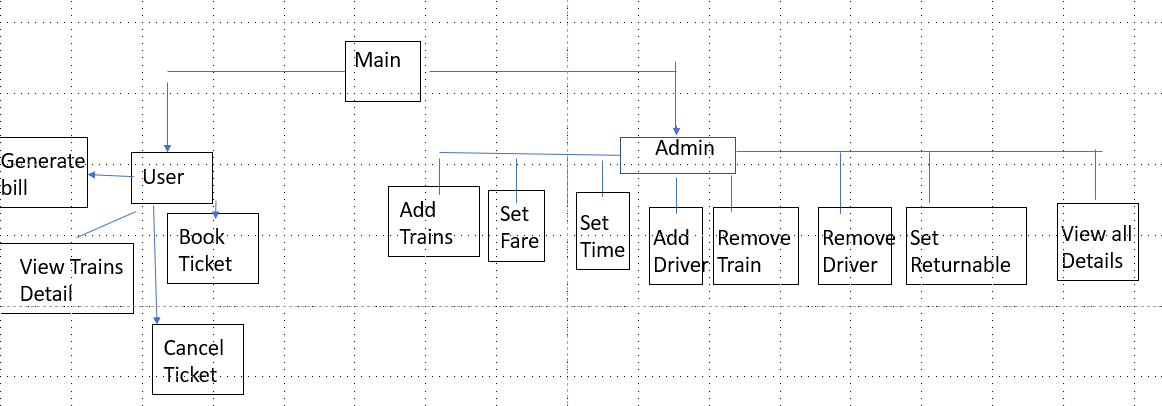
****

**Figure 5: Ticket canceled by User**

**Important Instructions**

* **Data Structures (Parallel Arrays)**
  + string roles[20]; // Array to store Registered ID Roles as Admin/User
  + string userName[20]; // Array to store UserName
  + string passwords[20]; // Array to store PAssword
  + string trainName[20]; // Array to store Registered Train Names
  + string trainGoingFrom[20]; // Array to store Departure city of Train
  + string trainGoingTo[20]; // Array to store Arraival city of Train
  + int trainCount = 0; // Global Veriable to store Number of Trains
  + string acfare[20]; // Array to store AC Class Fare
  + string standardfare[20]; // Array to store Standard Class Fare
  + string economyfare[20]; // Array to store Economy Class Fare
  + string trainTime[20]; // Array to store Train Time
  + string driverName[20]; // Array to store driver Name
  + string driverID[20]; // Array to store ID of Driver
  + int refund[3]; // Array to store Returnable ammount set by Admin
  + int tickets[100]; // number of tickets
  + int fare[100]; //stores Fare
  + string useNameWhenBuyTicket[100]; // When User Buy Ticket that name is used
  + string useIDWhenBuyTicket[100]; // When User Buy Ticket that ID which is his password is used
* **Function Prototypes**
* void topHeader(); // Thats Top Header which Display at screen
* void clearScreen(); // Use to Clear Screen
* int loginMenu(); // It returns a option selected from loginMenu
* bool checkIntegerInMenu(string checkOption); // Validty for integer in Menu
* void deleteLoginID(); // Use to Delete any User/Admin account
* void storeAfterDeleteLoginData(); // use To store ID name and password in file after deleting any
* string signIn(string name, string password); // It checks whether a enterd ID/Password Exist
* void subMenuBeforeMainMenu(string submenu); // It display Mainmenu
* void signInProcess(); // It takes ID names and Passwords To Login an Admin/User
* void signUpProcess(); // It takes Id names,passwords and Roles to Creat a account
* int adminMenu(); // It returns a option from Admin Menu
* int userMenu(); // It returns a option from user Menu
* void gotoxy(int x, int y); // use to Change Position of Any output
* void adminMenuProcess();
* void userMenuProcess();
* string getData(string line, int field); // use to get Data from any given file by comma seperating methode
* void storeLoginData(string users, string passwords, string roles); // use to store ID name and Passwords in file
* bool signUp(string name, string password, string role); // use for Validity of signUp
* void loadLoginData(); // use to load ID name and Passwords from file
* void storeTrainData(); // use to store train data in file
* void loadTrainData(); // use to load train Data from file
* void storeTrainFare(int Fareindex); // use to store train Fare in file
* void loadTrainFare(); // Use to load Train Fare from file
* void addTrain(); // use to Register a Train
* bool stringChecker(string number1); // use for Validty of string
* bool checkAddTrain(string checkName); // use to check Train Name already exist or not
* void setfare(); // use to set Registered Train Fare
* bool checkSetFare(int number2); // use to check train fare already set or not
* void setTime(); // use to set train time
* bool checkTrainTime(int number3); // use to check train time already set or not
* void updateTrainTiming(); // use to update Train Timing
* bool checkTrainTimeIfAlreadyExist(int number4);
* void deleteTrain(); // use To Unregister A train
* void addDriver(); // use to add driver to a registered Train
* bool checkAddDriver(int number); // use to check a train has driver or not
* void updateDriver(); // use to Update driver Details
* bool checkUpdateDriverIfAlreadyExist(string number5); // to check entered train has already driver or not
* void deleteDriver(); // Use to delete driver details
* void viewTrains(); // To view all Train Details
* void returnablePrice(); // To set Returnable ammount if Passenger Cancel a ticket
* void registerdTrain();
* void seeAvailableTrains();
* void seeTrainsfare();
* void seeTrainsTime();
* void bookTicket(); // It is use to book a registered Train Ticket
* void cancelTicket(); // It is use to cancel any booked Ticket
* void storeAfterDelete(); // It stores After Delete
* void storeTrainTime(int trainTimeIndex); // It stores Train Time
* void loadTrainTime(); // It Loads Train stored Time
* void storeDriverData(int driverDataIndex); // It store Driver Details
* void loadDriverData(); // It Loads Driver stored Details
* void storeAfterChangeDriver(); // It stores driver details after update
* void storeAfterChangeTime(); // It stores time after update
* bool integerChecker(string number); // It checks Validty of Integer in any String
* bool integerFareChecker(string number); // It Limits Fare Upto 5000
* bool checkDataOfTime(string checkIt); // It checks validty of Time
* bool returnAbleChecker(string returnable); // It checks Validy of integers in returnable amount entered
* void storeReturnable(); // It stores Returnable ammount
* void loadReturnable(); // It Loads stored returnable Amount
* void seeSoldTickets();
* bool checkFairIfExist(int number5); // It checks Wheter a entered user has bought any ticket or not
* void generateInvoiceBill(); // It Generates Bill of Passenger
* void storeTicketsData(int number6);
* void loadTicketData();
* void red(); //to color red
* void white(); //to color white
* void green(); //to color green
* void blue(); //to color blue

**Functions Working Flow**.



* **Complete Code of the Business Application**

// Libraries

#include <iostream>

#include <sstream>

#include <fstream>

#include <conio.h>

#include <windows.h>

#include <time.h>

using namespace std;

// ProtoTypes

void topHeader(); // Thats Top Header which Display at screen

void clearScreen(); // Use to Clear Screen

int loginMenu(); // It returns a option selected from loginMenu

bool checkIntegerInMenu(string checkOption); // Validty for integer in Menu

void deleteLoginID(); // Use to Delete any User/Admin account

void storeAfterDeleteLoginData(); // use To store ID name and password in file after deleting any

string signIn(string name, string password); // It checks whether a enterd ID/Password Exist

void subMenuBeforeMainMenu(string submenu); // It display Mainmenu

void signInProcess(); // It takes ID names and Passwords To Login an Admin/User

void signUpProcess(); // It takes Id names,passwords and Roles to Creat a account

int adminMenu(); // It returns a option from Admin Menu

int userMenu(); // It returns a option from user Menu

void gotoxy(int x, int y); // use to Change Position of Any output

void adminMenuProcess();

void userMenuProcess();

string getData(string line, int field); // use to get Data from any given file by comma seperating methode

void storeLoginData(string users, string passwords, string roles); // use to store ID name and Passwords in file

bool signUp(string name, string password, string role); // use for Validity of signUp

void loadLoginData(); // use to load ID name and Passwords from file

void storeTrainData(); // use to store train data in file

void loadTrainData(); // use to load train Data from file

void storeTrainFare(int Fareindex); // use to store train Fare in file

void loadTrainFare(); // Use to load Train Fare from file

void addTrain(); // use to Register a Train

bool stringChecker(string number1); // use for Validty of string

bool checkAddTrain(string checkName); // use to check Train Name already exist or not

void setfare(); // use to set Registered Train Fare

bool checkSetFare(int number2); // use to check train fare already set or not

void setTime(); // use to set train time

bool checkTrainTime(int number3); // use to check train time already set or not

void updateTrainTiming(); // use to update Train Timing

bool checkTrainTimeIfAlreadyExist(int number4);

void deleteTrain(); // use To Unregister A train

void addDriver(); // use to add driver to a registered Train

bool checkAddDriver(int number); // use to check a train has driver or not

void updateDriver(); // use to Update driver Details

bool checkUpdateDriverIfAlreadyExist(string number5); // to check entered train has already driver or not

void deleteDriver(); // Use to delete driver details

void viewTrains(); // To view all Train Details

void returnablePrice(); // To set Returnable ammount if Passenger Cancel a ticket

void registerdTrain();

void seeAvailableTrains();

void seeTrainsfare();

void seeTrainsTime();

void bookTicket(); // It is use to book a registered Train Ticket

void cancelTicket(); // It is use to cancel any booked Ticket

void storeAfterDelete(); // It stores After Delete

void storeTrainTime(int trainTimeIndex); // It stores Train Time

void loadTrainTime(); // It Loads Train stored Time

void storeDriverData(int driverDataIndex); // It store Driver Details

void loadDriverData(); // It Loads Driver stored Details

void storeAfterChangeDriver(); // It stores driver details after update

void storeAfterChangeTime(); // It stores time after update

bool integerChecker(string number); // It checks Validty of Integer in any String

bool integerFareChecker(string number); // It Limits Fare Upto 5000

bool checkDataOfTime(string checkIt); // It checks validty of Time

bool returnAbleChecker(string returnable); // It checks Validy of integers in returnable amount entered

void storeReturnable(); // It stores Returnable ammount

void loadReturnable(); // It Loads stored returnable Amount

void seeSoldTickets();

bool checkFairIfExist(int number5); // It checks Wheter a entered user has bought any ticket or not

void generateInvoiceBill(); // It Generates Bill of Passenger

void storeTicketsData(int number6);

void loadTicketData();

void red(); //to color red

void white(); //to color white

void green(); //to color green

void blue(); //to color blue

// Global Veriables

int userArrSize = 20; // Number of User Who can use That application

string roles[20]; // Array to store Registered ID Roles as Admin/User

string userName[20]; // Array to store UserName

string passwords[20]; // Array to store PAsswords

int usersCount = 0;

string trainName[20]; // Array to store Registered Train Names

string trainGoingFrom[20]; // Array to store Departure city of Train

string trainGoingTo[20]; // Array to store Arraival city of Train

int trainNumber = 0;

int trainCount = 0; // Global Veriable to store Number of Trains

string acfare[20]; // Array to store AC Class Fare

string standardfare[20]; // Array to store Standard Class Fare

string economyfare[20]; // Array to store Economy Class Fare

string trainTime[20]; // Array to store Train Time

string driverName[20]; // Array to store driver Name

string driverID[20]; // Array to store ID of Driver

int refund[3]; // Array to store Returnable ammount set by Admin

int tickets[100]; // number of tickets

int fare[100];

string useNameWhenBuyTicket[100]; // When User Buy Ticket that name is used

string useIDWhenBuyTicket[100]; // When User Buy Ticket that ID which is his password is used

string removedValue = "-";

int ticketData = 0;

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE); //Its to give colors

main()

{

clearScreen();

topHeader();

gotoxy(25,18);

cout << "Welcome!" << endl;

gotoxy(22,20);

cout << "Press any key to continue" << endl;

getch();

loadLoginData();

loadTrainData();

loadTrainFare();

loadTrainTime();

loadDriverData();

loadReturnable();

loadTicketData();

int loginOption = 0;

while (loginOption != 4)

{

clearScreen();

topHeader();

subMenuBeforeMainMenu("Login");

loginOption = loginMenu();

if (loginOption == 1)

{

signInProcess();

}

else if (loginOption == 2)

{

signUpProcess();

}

else if (loginOption == 3)

{

deleteLoginID();

}

refund[0] = 100; // by Default value of returnable percentage but Admin can change it

refund[1] = 75;

refund[2] = 50;

clearScreen();

}

cout << "press any key to continue!" << endl;

getch();

}

void deleteLoginID()

{

string removeID;

string removePassword;

cout << "Enter UserName:";

cin.clear();

cin.sync();

getline(cin, removeID);

bool isValid1 = stringChecker(removeID);

if (isValid1)

{

cout << "Enter your Password: " << endl;

cin.clear();

cin.sync();

getline(cin, removePassword);

bool flag = false;

for (int i = 0; i < 20; i++)

{

if (removeID == userName[i] && removePassword == passwords[i])

{

userName[i] = "-";

passwords[i] = "-";

roles[i] = "-";

flag = true;

storeAfterDeleteLoginData();

}

}

if (flag == true)

{

cout << "Deleted succesfully!" << endl;

}

else

{

cout << "Please Enter correct username/password!" << endl;

}

cout << "Press any key to continue";

getch();

}

}

void storeAfterDeleteLoginData()

{

fstream file;

file.open("signupData.txt", ios::out);

for (int x = 0; x < usersCount; x++)

{

file << userName[x] << "," << passwords[x] << "," << roles[x] << endl;

}

file.close();

}

void signUpProcess()

{

clearScreen();

string name;

string password;

string role;

topHeader();

subMenuBeforeMainMenu("SignUp");

cout << "Enter your Name: " << endl;

cin.clear();

cin.sync();

getline(cin, name);

bool isValid1 = stringChecker(name);

if (isValid1)

{

cout << "Enter your Password: " << endl;

cin.clear();

cin.sync();

getline(cin, password);

cout << "Enter your Role (Admin or User): " << endl;

cin.clear();

cin.sync();

getline(cin, role);

if (role == "Admin" || role == "User")

{

bool isValid3 = signUp(name, password, role);

if (isValid3)

{

cout << "SignedUp Successfully" << endl;

storeLoginData(name, password, role);

}

else

{

cout << "User name/Admin Role Already Exit! Try again" << endl;

}

}

else

{

cout << "Please Enter Admin/User! Try again" << endl;

}

}

else

{

cout << "Please Enter from A to Z! Try again" << endl;

}

cout << "Press any key To continue";

getch();

}

void signInProcess()

{

loadLoginData();

system("cls");

topHeader();

subMenuBeforeMainMenu("SignIn");

string name;

string password;

string role;

cout << "Enter your Name: " << endl;

cin.clear();

cin.sync();

getline(cin, name);

cout << "Enter your Password: " << endl;

cin.clear();

cin.sync();

getline(cin, password);

role = signIn(name, password);

if (role == "Admin")

{

cout << "Welcome As an Admin!" << endl;

adminMenuProcess();

}

else if (role == "User")

{

cout << "Welcome As a User!" << endl;

useNameWhenBuyTicket[ticketData] = name;

useIDWhenBuyTicket[ticketData] = password;

userMenuProcess();

}

else if (role != "User" || role != "Admin")

{

cout << "You Provided wrong Information " << endl;

}

cout << "press any key to continue!" << endl;

getch();

}

int loginMenu()

{

blue();

int choice;

string option;

cout << "Press 1. Already Reistered>> SignIn " << endl;

cout << "Press 2. Register Yourself>> SignUp " << endl;

cout << "Press 3. Delete Account >>" << endl;

cout << "Press 4. Exit " << endl;

cout << endl;

cout << "Enter the Option Number > ";

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = checkIntegerInMenu(option);

if (isValid1)

{

choice = stoi(option);

if (choice > 0 && choice < 5)

{

return choice;

}

else

{

cout << "Wrong Number Entered!" << endl;

}

}

else

{

cout << "Please Enter only Integer!" << endl;

}

cout << "Press any Key to Continue" << endl;

getch();

}

bool checkIntegerInMenu(string checkOption)

{

bool flag = false;

for (int x = 0; x < checkOption.length(); x++) // because in login only one x value less than 4 required

{

if (checkOption[x] > 48 && checkOption[x] < 58) // In ascii characters numbers start from 48 and end on 57

{

flag = true;

}

return flag;

}

}

string signIn(string name, string password)

{

for (int index = 0; index < usersCount; index++)

{

if (userName[index] == name && passwords[index] == password)

{

return roles[index];

}

}

return "undefined";

}

void loadLoginData()

{

string line;

fstream file;

file.open("signupData.txt", ios::in);

while (getline(file, line))

{

userName[usersCount] = getData(line, 1);

passwords[usersCount] = getData(line, 2);

roles[usersCount] = getData(line, 3);

usersCount++;

}

}

string getData(string line, int field)

{

int commacount = 1;

string item;

for (int idx = 0; idx < line.length(); idx++)

{

if (line[idx] == ',')

{

commacount++;

}

else if (commacount == field)

{

item = item + line[idx];

}

}

return item;

}

void storeLoginData(string users, string passwords, string roles)

{

bool flag = false;

for (int x = 0; x < userArrSize; x++)

{

if (users != userName[x])

{

flag = true;

}

}

if (flag)

{

fstream file;

file.open("signupData.txt", ios::app);

file << users << ',' << passwords << ',' << roles << endl;

file.close();

}

}

bool signUp(string name, string password, string role)

{

bool flag = true;

for (int x = 0; x < 20; x++)

{

if (role == roles[x] && role == "Admin")

{

flag = false;

}

else if (name == userName[x] && role == "User")

{

flag = false;

}

}

return flag;

}

bool checkAddTrain(string checkName)

{

bool flag = true;

for (int x = 0; x < trainCount; x++)

{

if (checkName == trainName[x])

{

flag = false;

}

}

return flag;

}

void subMenuBeforeMainMenu(string submenu)

{ green();

string message = submenu + " Menu";

cout << message << endl;

cout << "---------------------" << endl;

}

void subMenu(string submenu)

{ green();

string message = "Main Menu > " + submenu;

cout << message << endl;

cout << "---------------------" << endl;

}

void clearScreen()

{

system("cls");

}

void topHeader()

{

green();

cout <<" Today Date " \_\_DATE\_\_ << endl;

red();

cout << " \_\_\_\_\_\_ " << endl;

cout << " .-````..\_`. \_,## " << endl;

cout << "████████ ███  ███ ███████  \_..\_\_ |.-```-.| | \_,##``-.\_ " << endl;

cout << "   ██    ████  ████ ██       (\_\_\_\_\_)||\_\_\_\_\_|| |\_,##``-.\_,##`` " << endl;

cout << " ██  ██ ████ ██ ███████  \_| |.;-``-. | |#``-.\_,##`` " << endl;

cout << " ██  ██  ██  ██      ██  \_.;\_ `--` `\\ \\ |.``\\.\_,##`` " << endl;

cout << " ██  ██      ██ ███████  /.-.\\ `\\ |.-`:.`\_, |##`` " << endl;

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

cout << " `.\_\_\_\_.`\_.-`)\\--` /`-`` " << endl;

cout << " //||\\\\(\_.-`\_,`-`` " << endl;

cout << " (`-...-`)\_,##`` " << endl;

cout << " \_,##`-..,-:##` " << endl;

cout << "######################################################################################################" << endl;

}

int adminMenu()

{

system("cls");

topHeader();

subMenu("AdminMenu");

string option;

int choice;

blue();

cout << "Press 1.Add Trains" << endl;

cout << "Press 2.Set fare" << endl;

cout << "Press 3.set Train Timming" << endl;

cout << "Press 4.Update Train Time" << endl;

cout << "Press 5.remove train" << endl;

cout << "Press 6.Add Driver details" << endl;

cout << "Press 7.Update Driver details" << endl;

cout << "Press 8.remove Driver details" << endl;

cout << "Press 9.View all trains Detail" << endl;

cout << "Press 10.Set Rules to cancel a ticket" << endl;

cout << "Press 11.Exit " << endl;

cout << " enter your option:" << endl;

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = checkIntegerInMenu(option);

if (isValid1)

{

choice = stoi(option);

if (choice > 0 && choice < 13)

{

return choice;

}

else

{

cout << "Wrong Number Entered!" << endl;

}

}

else

{

cout << "Please Enter only Integer!" << endl;

}

cout << "Press any key to Continue" << endl;

getch();

}

void adminMenuProcess()

{

int adminMenuOption = 0;

while (adminMenuOption != 11)

{

adminMenuOption = adminMenu();

if (adminMenuOption == 1)

{

addTrain();

}

if (adminMenuOption == 2)

{

setfare();

}

if (adminMenuOption == 3)

{

setTime();

}

if (adminMenuOption == 4)

{

updateTrainTiming();

}

if (adminMenuOption == 5)

{

deleteTrain();

}

if (adminMenuOption == 6)

{

addDriver();

}

if (adminMenuOption == 7)

{

updateDriver();

}

if (adminMenuOption == 8)

{

deleteDriver();

}

if (adminMenuOption == 9)

{

viewTrains();

}

if (adminMenuOption == 10)

{

returnablePrice();

}

}

}

int userMenu()

{

clearScreen();

topHeader();

subMenu("userMenu");

string option;

int choice;

blue();

cout << "Press 1.See All available Trains" << endl;

cout << "Press 2.See Train fares" << endl;

cout << "Press 3.See Train Timmings" << endl;

cout << "Press 4.Book Tickets" << endl;

cout << "Press 5.Generate Invoice Bill" << endl;

cout << "Press 6.cancel tickets" << endl;

cout << "Press 7.Exit" << endl;

cout << " enter your Option" << endl;

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = checkIntegerInMenu(option);

if (isValid1)

{

choice = stoi(option);

if (choice > 0 && choice < 9)

{

return choice;

}

else

{

cout << "Wrong Number Entered!" << endl;

}

}

else

{

cout << "Please Enter only Integer!" << endl;

}

}

void userMenuProcess()

{

int userMenuOption = 0;

while (userMenuOption != 7)

{

userMenuOption = userMenu();

if (userMenuOption == 1)

{

seeAvailableTrains();

}

if (userMenuOption == 2)

{

seeTrainsfare();

}

if (userMenuOption == 3)

{

seeTrainsTime();

}

if (userMenuOption == 4)

{

bookTicket();

}

if (userMenuOption == 5)

{

generateInvoiceBill();

}

if (userMenuOption == 6)

{

cancelTicket();

}

}

cout << "Thank You For Using our service!" << endl;

}

void addTrain()

{

clearScreen();

topHeader();

green();

subMenu("Add\_Train");

string checkTrain;

string checkRoute1;

string checkRoute2;

blue();

registerdTrain();

cout << endl;

cout << " Enter Train " << trainCount + 1 << " name : " << endl;

cin.clear();

cin.sync();

getline(cin, checkTrain);

bool isValid1 = stringChecker(checkTrain);

if (isValid1 == true)

{

bool isValid2 = checkAddTrain(checkTrain);

if (isValid2 == true)

{

trainName[trainCount] = checkTrain;

cout << " Enter Going from city:" << trainCount + 1 << " : " << endl;

cin.clear();

cin.sync();

getline(cin, checkRoute1);

bool isValid3 = stringChecker(checkRoute1);

if (isValid3)

{

trainGoingFrom[trainCount] = checkRoute1;

cout << " Enter Going to city:" << trainCount + 1 << " : " << endl;

cin.clear();

cin.sync();

getline(cin, checkRoute2);

bool isValid4 = stringChecker(checkRoute2);

if (isValid4)

{

trainGoingTo[trainCount] = checkRoute2;

storeTrainData();

trainCount++;

cout << "Train Name Added Succesfully!" << endl;

}

else

{

cout << "Please only Enter Alphabets! Try Again" << endl;

}

}

else

{

cout << "Please only Enter Alphabets! Try Again" << endl;

}

}

else if (isValid2 == false)

{

cout << "Train Name Already exist" << endl;

}

}

else

{

cout << "Please only Enter Alphabets! Try Again" << endl;

}

cout << "Press any key to continue";

getch();

}

bool stringChecker(string number1)

{

bool flag = false;

for (int x = 0; x < number1.length(); x++)

{

if (number1[x] > 64 && number1[x] < 91 || number1[x] > 96 && number1[x] < 123 || number1[x] == ' ')

{

flag = true;

}

else

{

flag = false;

break;

}

}

return flag;

}

void storeTrainData()

{

fstream file;

file.open("TrainData.txt", ios::app);

file << trainName[trainCount] << ',' << trainGoingFrom[trainCount] << "," << trainGoingTo[trainCount] << endl;

file.close();

}

void loadTrainData()

{

string line;

fstream file;

file.open("trainData.txt", ios::in);

while (getline(file, line))

{

trainName[trainCount] = getData(line, 1);

trainGoingFrom[trainCount] = getData(line, 2);

trainGoingTo[trainCount] = getData(line, 3);

trainCount++;

}

file.close();

}

void setfare()

{

clearScreen();

topHeader();

green();

subMenu("setfare");

red();

registerdTrain();

string option;

cout << "Press Key accordingly To Set Train fare :" << endl;

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = integerChecker(option);

if (isValid1 == true)

{

int changer = stoi(option);

bool isValid2 = checkSetFare(changer);

if (isValid2 == true)

{

cout << "Train " << changer << " name is" << trainName[changer - 1] << endl;

string acfareCheck;

string standardfareCheck;

string economyfareCheck;

cout << "enter AC class fare:";

cin.clear();

cin.sync();

getline(cin, acfareCheck);

bool isCorrect1;

bool isCorrect2;

bool isCorrect3;

isCorrect1 = integerFareChecker(acfareCheck);

if (isCorrect1)

{

acfare[changer - 1] = acfareCheck;

cout

<< "Enter Standard Class fare:";

cin.clear();

cin.sync();

getline(cin, standardfareCheck);

isCorrect2 = integerFareChecker(standardfareCheck);

if (isCorrect2)

{

standardfare[changer - 1] = standardfareCheck;

cout << "Enter Economy class fare:";

cin.clear();

cin.sync();

getline(cin, economyfareCheck);

isCorrect3 = integerFareChecker(economyfareCheck);

if (isCorrect3)

{

economyfare[changer - 1] = economyfareCheck;

}

else

{

cout << "Please enter only integer in economy Fair !";

}

}

else

{

cout << "Please enter only integer in Bussiness class Fare!" << endl;

}

}

else

{

cout << "Please enter only integer in acFare!" << endl;

}

if (isCorrect1 && isCorrect2 && isCorrect3)

{

storeTrainFare(changer - 1); // Store train fare to file

cout << "Fare added Succesfully!" << endl;

}

}

else

{

cout << "That Train Fair Already set" << endl;

}

}

else

{

cout << "Please enter only integer! Try again " << endl;

}

cout << "Press any key to continue";

getch();

}

bool integerFareChecker(string number)

{

bool flag = false;

for (int x = 0; x < number.length(); x++)

{

if (number[x] > 47 && number[x] < 58) // In ascii characters numbers start from 48 and end on 57

{

int check = stoi(number);

if (check <= 5000) // You can add Fare upto 5000

{

flag = true;

}

}

return flag;

}

}

bool checkSetFare(int number2)

{

int checker = number2;

bool flag = false;

for (int x = 0; x < trainCount + 1; x++)

{

if (checker == x)

{

if (economyfare[x - 1].length() == 0 && acfare[x - 1].length() == 0 && standardfare[x - 1].length() == 0)

{

flag = true;

}

}

}

return flag;

}

void storeTrainFare(int Fareindex)

{

fstream file;

file.open("TrainFare.txt", ios::app);

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

{

if (Fareindex == i)

{

file << acfare[i] << "," << standardfare[i] << "," << economyfare[i] << endl;

}

}

file.close();

}

void loadTrainFare()

{

int fareCount = 0;

string line;

fstream file;

file.open("trainFare.txt", ios::in);

while (getline(file, line))

{

acfare[fareCount] = getData(line, 1);

standardfare[fareCount] = getData(line, 2);

economyfare[fareCount] = getData(line, 3);

fareCount++;

}

}

void registerdTrain()

{

blue();

cout << "Registered Trains>>" << endl;

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

{

cout << "train " << i + 1 << ". " << trainName[i] << " " << endl;

}

}

void setTime()

{

clearScreen();

topHeader();

green();

subMenu("SetTime");

string option;

registerdTrain();

cout << "You can only write in format like 4:PM" << endl

<< endl;

cout << "Enter Train Number to set Time:";

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = integerChecker(option);

if (isValid1 == true)

{

int changer = stoi(option);

bool isValid2 = checkTrainTime(changer);

if (isValid2 == true)

{

string trainTimeChecker;

cout << "Enter train " << option << " " << trainName[changer - 1] << " departure time:";

cin.clear();

cin.sync();

getline(cin, trainTimeChecker);

bool isCorrect1 = checkDataOfTime(trainTimeChecker);

if (isCorrect1 == true)

{

trainTime[changer - 1] = trainTimeChecker;

storeTrainTime(changer - 1);

cout << "Time Added Succesfully!" << endl;

}

else

{

cout << "Please Enter in Correct Format like 4:PM/AM Try Again" << endl;

}

}

else

{

cout << "Warning Already entered! Try again" << endl;

}

}

else

{

cout << "Please Enter Correct number" << endl;

}

cout << "Press any key to continue!";

getch();

}

bool checkDataOfTime(string checkIt)

{

bool flag = false;

for (int x = 0; x < checkIt.length(); x++) // In Ascii character 58=':' ,80='P',65='A',77='M'

{

if (checkIt[0] > 47 && checkIt[0] < 58 && checkIt[1] == 58 && (checkIt[2] == 80 || checkIt[2] == 65) && checkIt[3] == 77) // numbers start from 47

{

flag = true;

}

}

return flag;

}

void storeTrainTime(int trainTimeIndex)

{

fstream file;

file.open("TrainTime.txt", ios::app);

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

{

if (i == trainTimeIndex)

{

file << trainTime[i] << endl;

}

}

file.close();

}

void loadTrainTime()

{

int timeCount = 0;

string line;

fstream file;

file.open("trainTime.txt", ios::in);

while (getline(file, line))

{

trainTime[timeCount] = getData(line, 1);

timeCount++;

}

}

void updateTrainTiming()

{

clearScreen();

topHeader();

green();

subMenu("UpdateTrainTiming");

registerdTrain();

string option;

string trainTimeChecker;

cout << "Enter Train Number to set Time or update time:";

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = integerChecker(option);

if (isValid1)

{

int changer = stoi(option);

bool isValid2 = checkTrainTimeIfAlreadyExist(changer);

if (isValid2 == true)

{

cout << "Old time of train " << trainName[changer - 1] << " was:" << trainTime[changer - 1] << endl;

cout << "Enter New Time of train " << trainName[changer - 1] << " :" << endl;

cin.clear();

cin.sync();

getline(cin, trainTimeChecker);

bool isCorrect1 = checkDataOfTime(trainTimeChecker);

if (isCorrect1 == true)

{

trainTime[changer - 1] = trainTimeChecker;

storeAfterChangeTime();

cout << "Time Updated Succesfully!" << endl;

}

else

{

cout << "Please Enter in Correct Format like 4:PM/AM Try Again" << endl;

}

}

else

{

cout << "that Train Timing not exist! Please Add it first" << endl;

}

}

else

{

cout << "Please Enter Correct number" << endl;

}

cout << "Press any key to continue!";

getch();

}

bool checkTrainTimeIfAlreadyExist(int number4)

{

int checker = number4;

bool flag = true;

for (int x = 1; x < trainCount + 1; x++)

{

if (checker == x)

{

if (trainTime[x - 1].length() == 0)

{

flag = false;

}

}

}

return flag;

}

void storeAfterChangeTime()

{

fstream file;

file.open("trainTime.txt", ios::out);

for (int idx = 0; idx < trainCount; idx++)

{

file << trainTime[idx] << endl;

}

file.close();

}

void addDriver()

{

clearScreen();

topHeader();

green();

subMenu("Add\_Driver");

string option;

registerdTrain();

cout << "Enter Train Number to add driver:" << endl;

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = integerChecker(option);

if (isValid1)

{

int changer = stoi(option);

bool isValid2 = checkAddDriver(changer - 1);

if (isValid2 == true)

{

cout << "Enter " << trainName[changer - 1] << " Driver Name :" << endl;

cin.clear();

cin.sync();

getline(cin, driverName[changer - 1]);

cout << "Enter ID card of Driver having " << trainName[changer - 1] << " :";

cin.clear();

cin.sync();

getline(cin, driverID[changer - 1]);

storeDriverData(changer - 1);

cout << "Driver added succesfully" << endl;

}

else

{

cout << "Entered Already Exit! Try Again" << endl;

}

}

else

{

cout << "Please Enter integer only! Try Again" << endl;

}

cout << "press any key to continue";

getch();

}

bool checkAddDriver(int number)

{

int checker = number;

bool flag = true;

for (int x = 0; x < trainCount; x++)

{

if (checker == x)

{

if (driverName[x].length() != 0)

{

flag = false;

}

}

}

return flag;

}

void storeDriverData(int driverDataIndex)

{

fstream file;

file.open("DriverData.txt", ios::app);

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

{

if (i == driverDataIndex)

{

file << driverName[i] << "," << driverID[i] << endl;

}

}

file.close();

}

void loadDriverData()

{

int driverCount = 0;

string line;

fstream file;

file.open("driverData.txt", ios::in);

while (getline(file, line))

{

driverName[driverCount] = getData(line, 1);

driverID[driverCount] = getData(line, 2);

driverCount++;

}

}

void updateDriver()

{

clearScreen();

topHeader();

green();

subMenu("UpdateDriver");

registerdTrain();

string option;

string checkDriverName;

cout << "Enter Train Number to update driver Details:" << endl;

cin.clear();

cin.sync();

getline(cin, option);

bool isValid1 = integerChecker(option);

if (isValid1)

{

bool isValid2 = checkUpdateDriverIfAlreadyExist(option);

if (isValid2 == true)

{

int changer = stoi(option);

cout << "Old driver name of train " << trainName[changer - 1] << " was:" << driverName[changer - 1] << endl;

cout << "Old driver ID of train " << trainName[changer - 1] << " was:" << driverID[changer - 1] << endl;

cout << "Enter New driver name of train " << trainName[changer - 1] << " :" << endl;

cin.clear();

cin.sync();

getline(cin, checkDriverName);

bool isValid3 = stringChecker(checkDriverName);

if (isValid3)

{

driverName[changer - 1] = checkDriverName;

cout << "Enter new ID card of Driver having " << trainName[changer - 1] << " :" << endl;

cin.clear();

cin.sync();

getline(cin, driverID[changer - 1]);

storeAfterChangeDriver();

cout << "Driver Details updated succesfully";

}

else

{

cout << "Please Enter Alphabets only! Try again" << endl;

}

}

else

{

cout << "That Train has no existing Driver! Please Add Driver First" << endl;

}

}

else

{

cout << "Please Enter Alphabets only! Try again" << endl;

}

cout << "press any key to continue";

getch();

}

bool checkUpdateDriverIfAlreadyExist(string number5)

{

int checker = stoi(number5);

bool flag = true;

for (int x = 1; x < trainCount + 1; x++)

{

if (checker == x)

{

if (driverName[x - 1].length() == 0)

{

flag = false;

}

}

}

return flag;

}

void storeAfterChangeDriver()

{

fstream file;

file.open("driverData.txt", ios::out);

for (int idx = 0; idx < trainCount; idx++)

{

file << driverName[idx] << "," << driverID[idx] << endl;

}

file.close();

}

void viewTrains()

{

clearScreen();

topHeader();

blue();

subMenu("View\_all\_Trains");

gotoxy(2, 15);

cout << "Sr.";

gotoxy(5, 15);

cout << "TrainName";

gotoxy(20, 15);

cout << "From";

gotoxy(35, 15);

cout << "To";

gotoxy(50, 15);

cout << "AcClassFare";

gotoxy(65, 15);

cout << "BussinesFare";

gotoxy(80, 15);

cout << "economyFare";

gotoxy(95, 15);

cout << "TrainTime";

gotoxy(110, 15);

cout << "DriverName";

gotoxy(125, 15);

cout << "DriverID";

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

{

gotoxy(2, 17 + i);

cout << i + 1;

gotoxy(5, 17 + i);

cout << trainName[i];

gotoxy(20, 17 + i);

cout << trainGoingFrom[i];

gotoxy(35, 17 + i);

cout << trainGoingTo[i];

gotoxy(50, 17 + i);

cout << acfare[i];

gotoxy(65, 17 + i);

cout << standardfare[i];

gotoxy(80, 17 + i);

cout << economyfare[i];

gotoxy(95, 17 + i);

cout << trainTime[i];

gotoxy(110, 17 + i);

cout << driverName[i];

gotoxy(125, 17 + i);

cout << driverID[i];

}

cout << endl

<< "Press any key to continue";

getch();

}

void gotoxy(int x, int y)

{

COORD coordinates;

coordinates.X = x;

coordinates.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coordinates);

}

void returnablePrice()

{

system("cls");

topHeader();

subMenu("Returnable\_Price");

string checkOption1;

string checkOption2;

string checkOption3;

bool isValid1;

bool isValid2;

bool isValid3;

cout << "Enter returnable within 1 Days of Train Departure in %:" << endl;

cin.clear();

cin.sync();

getline(cin, checkOption1);

isValid1 = returnAbleChecker(checkOption1);

if (isValid1)

{

refund[0] = stoi(checkOption1);

cout << "Enter Refundable within 3 Days of Train Departure in %:" << endl;

cin.clear();

cin.sync();

getline(cin, checkOption2);

isValid2 = returnAbleChecker(checkOption2);

if (isValid2)

{

refund[1] = stoi(checkOption2);

cout << "Enter Refundable within 7 Days of Train Departure in %:" << endl;

cin.clear();

cin.sync();

getline(cin, checkOption3);

isValid2 = returnAbleChecker(checkOption3);

if (isValid3)

{

refund[2] = stoi(checkOption3);

}

else

{

cout << "Try again!" << endl;

}

}

else

{

cout << "Try again!" << endl;

}

}

else

{

cout << "Try again!" << endl;

}

if (isValid1 == true && isValid2 == true && isValid3 == true)

{

storeReturnable();

cout << "Returnable Amount in % added Succesfully" << endl;

}

cout << "Press any key to continue";

getch();

}

void storeReturnable()

{

fstream file;

file.open("Returnable.txt", ios::out);

file << refund[0] << "\t" << refund[1] << "\t" << refund[2];

file.close();

}

void loadReturnable()

{

int AC;

int Bussiness;

int economy;

fstream file;

file.open("Returnable.txt", ios::in);

while (file.eof())

{

refund[0] = AC;

refund[1] = Bussiness;

refund[2] = economy;

}

file.close();

}

bool returnAbleChecker(string returnable)

{

bool flag = false;

for (int x = 0; x < returnable.length(); x++)

{

if (returnable[x] > 47 && returnable[x] < 58) // In ascii characters numbers start from 48 and end on 57

{

flag = true;

}

return flag;

}

}

void deleteTrain()

{

clearScreen();

topHeader();

green();

subMenu("Delete\_Train");

registerdTrain();

string checkRemoveTrain;

string removeTrain;

cout << "Enter Train Name To Remove it:";

cin.clear();

cin.sync();

getline(cin, checkRemoveTrain);

bool isValid1 = stringChecker(checkRemoveTrain);

if (isValid1)

{

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

{

if (checkRemoveTrain == trainName[i] )

{

for(int x=i;x<trainCount;x++)

{

trainName[x] = trainName[x+1];

trainGoingFrom[x] = trainGoingFrom[x + 1];

trainGoingTo[x] = trainGoingTo[x + 1];

trainTime[i] = trainTime[x + 1];

acfare[x] = acfare[x + 1];

standardfare[x] = standardfare[x + 1];

economyfare[x] = economyfare[x + 1];

}

}

}

trainCount--;

cout << "Train " << checkRemoveTrain << " has been removed!" << endl;

storeAfterDelete();

}

else

{

cout << "Please enter correct Train Name" << endl;

}

cout << "Press any key to continue";

getch();

}

void storeAfterDelete()

{

fstream file;

file.open("TrainData.txt", ios::out);

for (int idx = 0; idx < trainCount; idx++)

{

file << trainName[idx] << ',' << trainGoingFrom[idx] << "," << trainGoingTo[idx] << endl;

}

file.close();

}

void deleteDriver()

{

system("cls");

topHeader();

green();

subMenu("Delete\_Driver");

registerdTrain();

string check;

cout << "Enter Train Name to remove:" << endl;

cin.clear();

cin.sync();

getline(cin, check);

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

{

if (check == trainName[i])

{

driverName[i] = removedValue;

driverID[i] = removedValue;

}

}

storeAfterChangeDriver();

cout << "Driver Name removed succesfully!";

cout << "Press any key to continue";

getch();

}

bool integerChecker(string number)

{

bool flag = false;

for (int x = 0; x < number.length(); x++)

{

if (number[x] > 47 && number[x] < 58) // In ascii characters numbers start from 48 and end on 57

{

int check = stoi(number); // for values greater than 9 we have to check them with number of Trains

if (check < trainCount + 1)

{

flag = true;

}

}

return flag;

}

}

bool checkTrainTime(int number3)

{

int checker = number3;

bool flag = true;

for (int x = 1; x < trainCount + 1; x++)

{

if (checker == x)

{

if (trainTime[x - 1].length() != 0)

{

flag = false;

}

}

}

return flag;

}

// USER Menu Fnction definations

void seeAvailableTrains()

{

system("cls");

topHeader();

green();

subMenu("see\_available\_Trains");

cout << "All Available trains are listed below:" << endl;

for (int i = 0; i < trainNumber + trainCount; i++)

{

cout << "train " << i + 1 << " : " << trainName[i] << endl;

}

cout << "Press any key to continue";

getch();

}

void seeTrainsfare()

{

system("cls");

topHeader();

green();

subMenu("see\_Trains\_fares");

cout << "All Available trains fares are listed below:" << endl;

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

{

cout << "train " << i + 1 << " : " << trainName[i] << endl;

cout << "AC class fare of" << trainName[i] << " : " << acfare[i] << endl;

cout << "Standard class fare of" << trainName[i] << " : " << standardfare[i] << endl;

cout << "Economy class fare of" << trainName[i] << " : " << economyfare[i] << endl;

}

cout << "Press any key to continue";

getch();

}

void seeTrainsTime()

{

system("cls");

topHeader();

green();

subMenu("see\_Trains\_Timings");

cout << "All Available trains Timings are listed below:" << endl;

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

{

cout << "train " << i + 1 << " : " << trainName[i];

cout << " Time of Departure is: " << trainTime[i] << endl;

}

cout << "Press any key to continue";

getch();

}

void bookTicket()

{

clearScreen();

topHeader();

green();

subMenu("BOOK\_Ticket");

int trainNumberforTicket;

registerdTrain();

string check;

string checkClass;

string getClass;

string ticketCheck;

cout << "Enter Train Number who's Ticket You want to buy";

cin.clear();

cin.sync();

getline(cin, check);

bool isValid = integerChecker(check);

if (isValid == true)

{

trainNumberforTicket = stoi(check);

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

{

if (trainNumberforTicket - 1 == i)

{

cout << trainName[i] << " Going From " << trainGoingFrom[i] << " to " << trainGoingTo[i] << endl;

cout << "1.AC Fare>>" << acfare[i] << endl;

cout << "2.Standard Fare>>" << standardfare[i] << endl;

cout << "3.Economy Fare>>" << economyfare[i] << endl;

cout << "Enter a number for Travelling catogery:" << endl;

cin.clear();

cin.sync();

getline(cin, checkClass);

bool isValid2 = integerChecker(checkClass);

if (isValid2 == true)

{

int converter = stoi(checkClass);

if (converter > 0 && converter < 4)

{

getClass = checkClass;

cout << "Enter Number of tickets:" << endl;

cin.clear();

cin.sync();

getline(cin, ticketCheck);

bool isValid3 = integerChecker(ticketCheck);

if (isValid3 == true)

{

tickets[i + ticketData] = stoi(ticketCheck);

if (stoi(getClass) == 1) // 1 mean AC class

{

fare[i + ticketData] = tickets[i + ticketData] \* stoi(acfare[i]);

}

else if (stoi(getClass) == 2) // 2 mean Bussiness class

{

fare[i + ticketData] = tickets[i + ticketData] \* stoi(standardfare[i]);

}

else if (stoi(getClass) == 3) // 3 mean for economy class

{

fare[i + ticketData] = tickets[i + ticketData] \* stoi(economyfare[i]);

}

cout << "Ticket Booked Succesfully" << endl;

cout << "Your Fair is:" << fare[i + ticketData] << endl;

storeTicketsData(trainNumberforTicket - 1);

ticketData++;

}

else

{

cout << "Please Enter only Integer!" << endl;

}

}

else

{

cout << "Please Enter Valid Number!" << endl;

}

}

else

{

cout << "Please enter only Integer!" << endl;

}

}

}

}

else

{

cout << "Please Enter a number only! Try Again" << endl;

}

cout << "Press any key to continue";

getch();

}

void cancelTicket()

{

clearScreen();

topHeader();

green();

subMenu("Cancel\_Ticket");

registerdTrain();

string check;

cout << "Enter Train Number who's Ticket You want to cancel" << endl;

cin.clear();

cin.sync();

getline(cin, check);

bool isValid1 = integerChecker(check);

if (isValid1 == true)

{

int changer = stoi(check);

bool isValid2 = checkFairIfExist(changer - 1);

if (isValid2)

{

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

{

if (changer - 1 == i)

{

cout << "Train" << trainName[i] << "You Bought " << tickets[i] << " tickets of fare " << fare[i] << endl;

int daysDeparture;

string checkDepartureDays;

cout << " Enter after how many days you cancel ticket after train departure :" << endl;

cin.clear();

cin.sync();

getline(cin, checkDepartureDays);

bool isValid4 = integerChecker(checkDepartureDays);

if (isValid4 == true)

{

daysDeparture = stoi(checkDepartureDays);

if (daysDeparture < 1)

{

cout << "Returnable ammount is 100%" << endl;

fare[i + ticketData -1] = fare[i + ticketData-1];

}

else if (daysDeparture > 1 && daysDeparture < 3)

{

cout << "Returnable ammount is " << refund[0] << "%" << endl;

fare[i + ticketData-1] = (fare[i + ticketData-1] - fare[i + ticketData-1] \* (100 - refund[0]) \* 100);

}

else if (daysDeparture > 3 && daysDeparture < 7)

{

cout << "Returnable ammount is " << refund[1] << "%" << endl;

fare[i + ticketData -1] = (fare[i + ticketData-1] - (fare[i + ticketData-1]) \* (100 - refund[1]) \* 100);

}

else if (daysDeparture > 7)

{

cout << "Returnable ammount is " << refund[2] << "%" << endl;

fare[i + ticketData-1] = ((fare[i + ticketData-1]) - (fare[i + ticketData-1]) \* (100 - refund[2]) \* 100);

}

cout << "Ticket Canceled and your Money Returnable is " << fare[i + ticketData -1] << endl;

fare[i + ticketData -1] = 0;

tickets[i + ticketData -1] = 0;

}

}

}

}

else

{

cout << "You Have Not Bought Any Ticket!" << endl;

}

}

else

{

cout << "PLease Enter a number! Try Again" << endl;

}

cout << "Press any key to continue";

getch();

}

bool checkFairIfExist(int number5)

{

bool flag = false;

for (int x = 0; x < trainCount; x++)

{

if (number5 == x && (fare[x + ticketData-1]) != 0)

{

flag = true;

}

}

return flag;

}

void generateInvoiceBill()

{

clearScreen();

topHeader();

green();

subMenu("Generate\_Bill");

int trainNumberforTicket;

registerdTrain();

string ticketCheck;

cout << "Enter Train Number who's Ticket You want to buy";

cin.clear();

cin.sync();

getline(cin, ticketCheck);

bool isValid = integerChecker(ticketCheck);

if (isValid == true)

{

int changer = stoi(ticketCheck);

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

{

if (changer - 1 == i)

{

if ((fare[i+ticketData -1]) != 0)

{ green();

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "| \* \* \* \* \* \* \* \* \*" << endl;

cout << "| Dep.Date:" << \_\_DATE\_\_ << " Dep.Time:" << trainTime[i] << " " << endl;

cout << "| " << endl;

cout << "| Train Name: " << trainName[i] << " From " << trainGoingFrom[i] << " to " << trainGoingTo[i] << endl;

cout << "| Diver Name:" << driverName[i] << " Driver ID:" << driverID[i] << endl;

cout << "| Passenger Name:" << useNameWhenBuyTicket[ticketData-1] << " Passenger ID:" << useIDWhenBuyTicket[ticketData-1] << endl;

cout << "| Total Rs:" << fare[i + ticketData -1] << endl;

cout << "| Book Time:" \_\_TIME\_\_ << endl;

cout << "| Sr.NO " << i + ticketData << endl;

cout << "|" << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

else

{

cout << "Book Ticket First!" << endl;

}

}

}

}

else

{

cout << "Plese Enter only Integer!" << endl;

}

cout << "Press any key to Continue" << endl;

getch();

}

void storeTicketsData(int number6)

{

fstream file;

file.open("passengerData.txt", ios::app);

for (int x = 0; x < 100; x++)

{

if (number6 == x)

{

file << fare[x + ticketData] << "\t" << useNameWhenBuyTicket[ticketData] << "\t" << useIDWhenBuyTicket[ticketData] << endl;

}

}

file.close();

}

void loadTicketData()

{

int passengerFare;

string user;

string password;

fstream file;

file.open("passengerData.txt", ios::in);

for (int x = 0; x < ticketData; x++)

{

while (file.eof())

{

fare[x] = passengerFare;

useNameWhenBuyTicket[x] = user;

useIDWhenBuyTicket[x] = password;

ticketData++;

}

}

file.close();

}

void red()

{

SetConsoleTextAttribute(h, 4);

}

void green()

{

SetConsoleTextAttribute(h, 10);

}

void blue()

{

SetConsoleTextAttribute(h, 3);

}

void white()

{

SetConsoleTextAttribute(h, 15);

}

* **Weakness in the Business Application**

I can Not control it by arrow keys.

* **Future Directions**
  + I want to add graphics in it .making a business application is my favourite.
  + I want to control it by cursor mouse.

**Student Reg. No. :**  2022-CS-78  **Student Name.**  Muhammad Tabish Akhtar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **A-Extensive Evidence** | **B-Convincing Evidence** | **C-Limited Evidence** | **D-No Evidence** |
| Documentation Formatting  **Grade:** | All the documentation meets all the criteria. | Documentation is well formatted but some of the criteria is not fulfilled. | Documentation is required a lot of improvement. | Documentation is not Available |
| **Documentation Formatting Criteria:** In **Binder**, **Title** Page, **Header**-Footers, Font **Style**, Font **Size** all are all consistence and according to given **guidelines**. Project **Poster** is professionally design and well presented | | | | |
| Documentation Contents  **Grade:** | Documentation includes all of the criteria. | Documentation meet more than 80% of the criteria given. | Documentation meet more than 50% of the criteria. | When the documentation meet less than 50% of the criteria. |
| **Documentation Contents Criteria:** **Title** Page - **Table** of Contents - Project **Abstract** - **Functional** Requirements - **Wire** Frames –**Data Flow** Diagram-**Data** Structure (Arrays)-**Function** Headers and Description -Project **Code.** - **Weakness** in the Project and **Future** Directions. - **Conclusion** and What your **Learn** from the Project and Course and What is your **Future** Planning. | | | | |
| Project Complexity  **Grade:** | Project has at least 2 user’s types and each user has at least 5 functionalities. | Project complexity meet 80% criteria given in extensive evidence | Project complexity meet 50% criteria given in extensive evidence | Project complexity meet less than 50% criteria given in extensive evidence |
| Code Style  **Grade:** | All Code style criteria is followed | All code style criteria followed but some improvements required | lot of improvements required in coding style. | **Did not follow** code style, |
| **Code Style Criteria:**  Consistent code style. Code is well indented. Variable and Function names are well defined.  White Spaces are well used. Comments are added. | | | | |
| Code Documentation Mapping  **Grade:** | Code and documentation is synchronized. | Code and documentation does not synchronized at **some** places | Code and documentation does not synchronized at **many** places | Code and documentation **does not** synchronized. |
| Data Structure (Arrays)  **Grade:** | Data structure is sufficient for the project requirements | Data Structure is sufficient but require improvement to meet project requirements. | Data structure is not sufficient and need a lot of improvement | Data Structure is not properly identified and declared. |
| Modularity  **Grade:** | Meet all Modularity criteria | Meet all Modularity criteria but at some places it is missing | Do not sufficiently meet the modularity criteria. | No modularity or very minimum modularity. |
| **Modularity criteria:** Functions are defined for each major feature. Functions are independent (identify from parameter list and return types). | | | | |
| Validations  **Grade:** | Validations on all number type inputs are applied | Validations are applied but at some places it is missing. | Validations are missing at lot of places | No Validations are used |
| File Handling  **Grade:** | Separate files for separate data. Data in csv format | File handing require some improvements | File handing require a lot of improvements | Not implemented |
| Aesthetics of the User Interface  **Grade:** | UI is presentable. Proper coloring, Headers and clear screen is done | UI require some improvements | UI require a lot of improvements | Not implemented |
| Presentation and Demo  **Grade:** | Presentation and Demo was 100% working | Presentation and Demo require some improvements | Presentation and Demo require a lot of improvements | Presentation was not ok and Demo was not working |
| Student Understanding with the Code.  **Grade:** | Student has complete understanding how the code is working and knows the concept. | Student has good understand but some place he does not know the concepts | Student has a very little understand and lack the major concepts. | Student does not have any level of understanding of the code. |

|  |  |
| --- | --- |
| **Checked by:** |  |
| **Comments:** |  |