#include <iostream>

#include <conio.h>

#include <stdlib.h>

#include <fstream>

#include <string.h>

#include <vector>

#include <map>

#include <sstream>

#include <sstream>

#ifdef \_WIN32

#include <windows.h> // For Sleep function on Windows

#else

#include <unistd.h> // For sleep function on Unix-like systems

#endif

using namespace std;

class login

{

public:

int loginsys();

};

typedef struct vehicle

{

string drivername;

string vehiclename;

string vehicleid;

string ownerName;

string contactNumber;

int timestay;

bool isBicycle;

bool isTwoWheeler;

bool requiresCharging;

string parkingslot;

string type;

int hoursParked;

}

vehicle;

typedef struct Reservation {

string vehicleId;

string driverName;

string parkingSlot;

string reservationTime; // You can use a more complex time structure if needed

} Reservation;

int login ::loginsys()

{

while (1)

{

cout << "\n\n\n\n\t\t\t\t----------> VEHICLE PARKING MANAGEMENT SYSTEM <----------" << endl;

cout << "\n\n\n\t\t\t\t\t Hey there! Ready to manage your parking?";

cout << "\t\t\t\t\t\t\nWelcome To Our Parking System \n\n";

cout<<"\t\t\t\t\t\Manage parking operations with ease and efficiency!\n";

cout<<"\t\t\t\t\t\ Developed with and for all parking needs\n";

system("PAUSE");

break;

}

return 1;

}

class User

{

public:

void onlineAuthorization();

std::string username;

std::string password;

bool isLoggedIn;

User(std::string user, std::string pass) : username(user), password(pass), isLoggedIn(false) {}

bool authenticate(std::string user, std::string pass) {

if (user == username && pass == password) {

isLoggedIn = true;

return true;

}

return false;

}

void logout() {

isLoggedIn = false;

}

};

class ParkingManagement

{

private:

User \*loggedInUser;

public:

void park(ParkingManagement);

void vehicledetail();

void addexpensedb(vehicle VEHICLE[], int);

void expense();

void removevehicle();

void getfeedback();

void selectParkingPlace(vehicle &v);

void viewAllParkedVehicles();

void editVehicleDetails();

void searchParkingPlace();

void viewAllFeedback();

void generateReport();

void addParkingSlot();

void viewAllParkingSlots();

void removeParkingSlot();

void allocateParkingSlot();

void viewParkingSlotDetails();

void editParkingSlotDetails();

void viewVehicleDetailsById();

void editExpense();

void deleteExpense();

double getParkingFee();

void listVehiclesByType();

string getMostFrequentParkingSlot();

void addVehicleHistory();

void viewVehicleHistory();

void sendReminderForPayment();

void generateExpenseReport();

bool viewParkingSlotAvailability();

void findParkingSlotByID();

void removevehiclee();

void viewStatistics();

void help();

void emergencyAlert();

void settings();

void adminMode();

void park();

void initializeAvailableSlots();

int checkAvailableSlots(string parkingPlace);

void updateAvailableSlots(string parkingPlace, bool isParking);

void searchByDriverName();

void displayParkingRules();

void notifyOverduePayment();

void viewAllVehiclesNeedingCharging();

void manageParkingSlots();

void addParkingPlace();

void reserveParkingSpot();

void viewAllReservedVehicles();

void CancelReservedVehicle();

void checkReservationStatus();

void modifyReservation();

};

int count = 0;

vector<string> parkingslots =

{

"Downtown Parking",

"Central Park Garage",

"City Hall Lot",

"Mall Parking",

"Airport Parking",

"Train Station Parking",

"Beach Parking",

"Stadium Parking",

"Library Parking",

"Museum Parking",

"Hospital Parking",

"University Parking",

"Shopping Center Parking",

"Residential Area Parking",

"Business District Parking",

"Cinema Parking",

"Zoo Parking",

"Park and Ride",

"Sports Complex Parking",

"Convention Center Parking",

"Hotel Parking",

"Theater Parking",

"Restaurant Parking",

"Night Club Parking",

"Market Parking",

"Church Parking",

"Temple Parking",

"Mosque Parking",

"School Parking",

"College Parking",

"Playground Parking",

"Community Center Parking",

"Fire Station Parking",

"Police Station Parking",

"Government Office Parking",

"Bus Terminal Parking",

"Taxi Stand Parking",

"Ferry Terminal Parking",

"Bridge Parking",

"Tunnel Parking",

"Court House Parking",

"Post Office Parking",

"Swimming Pool Parking",

"Gym Parking",

"Recreation Center Parking",

"Amusement Park Parking",

"Water Park Parking",

"Casino Parking",

"Race Track Parking",

"Botanical Garden Parking"

};

int totalSlots = sizeof(parkingslots) / sizeof(parkingslots[0]);

map<string, int> parkingFees =

{

{"Downtown Parking", 20},

{"Central Park Garage", 25},

{"City Hall Lot", 15},

{"Mall Parking", 30},

{"Airport Parking", 50},

{"Train Station Parking", 40},

{"Beach Parking", 35},

{"Stadium Parking", 45},

{"Library Parking", 10},

{"Museum Parking", 15},

{"Hospital Parking", 25},

{"University Parking", 20},

{"Shopping Center Parking", 30},

{"Residential Area Parking", 10},

{"Business District Parking", 35},

{"Cinema Parking", 25},

{"Zoo Parking", 20},

{"Park and Ride", 15},

{"Sports Complex Parking", 40},

{"Convention Center Parking", 35},

{"Hotel Parking", 30},

{"Theater Parking", 25},

{"Restaurant Parking", 20},

{"Night Club Parking", 30},

{"Market Parking", 15},

{"Church Parking", 10},

{"Temple Parking", 10},

{"Mosque Parking", 10},

{"School Parking", 10},

{"College Parking", 20},

{"Playground Parking", 15},

{"Community Center Parking", 10},

{"Fire Station Parking", 5},

{"Police Station Parking", 5},

{"Government Office Parking", 25},

{"Bus Terminal Parking", 20},

{"Taxi Stand Parking", 15},

{"Ferry Terminal Parking", 25},

{"Bridge Parking", 20},

{"Tunnel Parking", 20},

{"Court House Parking", 25},

{"Post Office Parking", 15},

{"Swimming Pool Parking", 15},

{"Gym Parking", 15},

{"Recreation Center Parking", 15},

{"Amusement Park Parking", 35},

{"Water Park Parking", 35},

{"Casino Parking", 50},

{"Race Track Parking", 45},

{"Botanical Garden Parking", 20}

};

map<string, int> chargingStationFees =

{

{"Downtown Parking", 10},

{"Central Park Garage", 15},

{"City Hall Lot", 5},

{"Mall Parking", 10},

{"Airport Parking", 20},

{"Train Station Parking", 15},

{"Beach Parking", 10},

{"Stadium Parking", 20},

{"Library Parking", 5},

{"Museum Parking", 5},

{"Hospital Parking", 10},

{"University Parking", 10},

{"Shopping Center Parking", 10},

{"Residential Area Parking", 5},

{"Business District Parking", 10},

{"Cinema Parking", 10},

{"Zoo Parking", 10},

{"Park and Ride", 5},

{"Sports Complex Parking", 15},

{"Convention Center Parking", 10},

{"Hotel Parking", 10},

{"Theater Parking", 10},

{"Restaurant Parking", 10},

{"Night Club Parking", 15},

{"Market Parking", 5},

{"Church Parking", 5},

{"Temple Parking", 5},

{"Mosque Parking", 5},

{"School Parking", 5},

{"College Parking", 10},

{"Playground Parking", 5},

{"Community Center Parking", 5},

{"Fire Station Parking", 5},

{"Police Station Parking", 5},

{"Government Office Parking", 10},

{"Bus Terminal Parking", 10},

{"Taxi Stand Parking", 5},

{"Ferry Terminal Parking", 15},

{"Bridge Parking", 10},

{"Tunnel Parking", 10},

{"Court House Parking", 10},

{"Post Office Parking", 5},

{"Swimming Pool Parking", 5},

{"Gym Parking", 5},

{"Recreation Center Parking", 5},

{"Amusement Park Parking", 15},

{"Water Park Parking", 15},

{"Casino Parking", 20},

{"Race Track Parking", 20},

{"Botanical Garden Parking", 10}

};

map<string, int> totalParkingSlots = {

{"Downtown Parking", 100},

{"Central Park Garage", 150},

{"City Hall Lot", 80},

{"Mall Parking", 200},

{"Airport Parking", 500},

{"Train Station Parking", 300},

{"Beach Parking", 120},

{"Stadium Parking", 400},

{"Library Parking", 50},

{"Museum Parking", 75},

{"Hospital Parking", 150},

{"University Parking", 250},

{"Shopping Center Parking", 300},

{"Residential Area Parking", 100},

{"Business District Parking", 200},

{"Cinema Parking", 100},

{"Zoo Parking", 150},

{"Park and Ride", 200},

{"Sports Complex Parking", 250},

{"Convention Center Parking", 300},

{"Hotel Parking", 150},

{"Theater Parking", 100},

{"Restaurant Parking", 80},

{"Night Club Parking", 100},

{"Market Parking", 120},

{"Church Parking", 100},

{"Temple Parking", 80},

{"Mosque Parking", 80},

{"School Parking", 100},

{"College Parking", 200},

{"Playground Parking", 50},

{"Community Center Parking", 80},

{"Fire Station Parking", 30},

{"Police Station Parking", 40},

{"Government Office Parking", 150},

{"Bus Terminal Parking", 100},

{"Taxi Stand Parking", 50},

{"Ferry Terminal Parking", 100},

{"Bridge Parking", 60},

{"Tunnel Parking", 40},

{"Court House Parking", 100},

{"Post Office Parking", 50},

{"Swimming Pool Parking", 80},

{"Gym Parking", 60},

{"Recreation Center Parking", 100},

{"Amusement Park Parking", 300},

{"Water Park Parking", 200},

{"Casino Parking", 400},

{"Race Track Parking", 300},

{"Botanical Garden Parking", 100}

};

vector<string> emergencySlots =

{

"Ambulance Parking",

"Fire Truck Parking",

"Police Vehicle Parking",

"Emergency Vehicle Parking"

};

map<string, int> availableParkingSlots;

void ParkingManagement::selectParkingPlace(vehicle &v)

{

int choice;

cout << "Available Parking Places: " << endl;

for (int i = 0; i < parkingslots.size(); ++i)

{

cout << i + 1 << ". " << parkingslots[i] << " (Fee: " << parkingFees[parkingslots[i]] << " per hour)" << endl;

}

cout << "Select a parking place by entering the corresponding number: ";

cin >> choice;

while (choice < 1 || choice > parkingslots.size())

{

cout << "Invalid choice. Please select a number between 1 and " << parkingslots.size() << ": ";

cin >> choice;

}

v.parkingslot = parkingslots[choice - 1];

}

void ParkingManagement ::park(ParkingManagement obj)

{

cout << " -------------> Vehicle Parking System <-------------" << endl

<< endl;

ofstream out;

vehicle VEHICLE[10];

out.open("vehicleparkdb.txt", ios::app);

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

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

cout << "Enter Your Name : ";

getchar();

getline(cin, VEHICLE[count].drivername);

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

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

cout << "Enter Your Vehicle Name : ";

getline(cin, VEHICLE[count].vehiclename);

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

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

cout << "Enter Your Vehicle Number Plate (You Can Enter A Unique Alphabet Sequence) (Remember It Well!) : ";

cin >> VEHICLE[count].vehicleid;

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

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

cout << "Enter Your Time Stay In Hours (Integer Only) : ";

cin >> VEHICLE[count].timestay;

char vehicleType;

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

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

cout << "Is it a bicycle? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isBicycle = (vehicleType == 'y' || vehicleType == 'Y');

if (!VEHICLE[count].isBicycle)

{

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

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

cout << "Is it a two-wheeler? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isTwoWheeler = (vehicleType == 'y' || vehicleType == 'Y');

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

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

cout << "Does it require charging? (y/n): ";

cin >> vehicleType;

VEHICLE[count].requiresCharging = (vehicleType == 'y' || vehicleType == 'Y');

}

else

{

VEHICLE[count].isTwoWheeler = false;

VEHICLE[count].requiresCharging = false;

}

selectParkingPlace(VEHICLE[count]);

out << "Vehicle Name : " << VEHICLE[count].vehiclename

<< ", Vehicle Id : " << VEHICLE[count].vehicleid << "\n"

<< "Driver Name : " << VEHICLE[count].drivername

<< ", Vehicle Time Stay : " << VEHICLE[count].timestay << " hours\n"

<< "Is Bicycle: " << (VEHICLE[count].isBicycle ? "Yes" : "No") << "\n"

<< "Is Two-Wheeler: " << (VEHICLE[count].isTwoWheeler ? "Yes" : "No") << "\n"

<< "Requires Charging: " << (VEHICLE[count].requiresCharging ? "Yes" : "No") << "\n"

<< "Parking Slot: " << VEHICLE[count].parkingslot << "\n";

addexpensedb(VEHICLE, count);

count++;

out.close();

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

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

cout << "Your Vehicle Is Parked Now!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

// Initialize available slots in the constructor or at program start

void ParkingManagement ::vehicledetail()

{

system("CLS");

string detailid;

string line;

bool found = false;

cout << "Enter the vehicle Id: ";

getchar();

getline(cin, detailid);

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

ifstream in;

in.open("vehicleparkdb.txt");

while (getline(in, line)) {

if (line.find(detailid) != string::npos)

{

cout << "Vehicle Found!" << endl;

cout << line << endl;

getline(in, line); // Read the next line.

cout << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "No vehicle found for the vehicle id: " << detailid << endl;

}

cout << "Press Enter Key To Return" << endl;

getchar();

}

void ParkingManagement::initializeAvailableSlots()

{

for (const auto& slot : totalParkingSlots)

{

availableParkingSlots[slot.first] = slot.second;

}

}

int ParkingManagement::checkAvailableSlots(string parkingPlace) {

if (availableParkingSlots.find(parkingPlace) != availableParkingSlots.end())

{

return availableParkingSlots[parkingPlace];

}

return -1; // Return -1 if parking place not found

}

// Function to update available slots when a vehicle is parked

void ParkingManagement::updateAvailableSlots(string parkingPlace, bool isParking)

{

if (availableParkingSlots.find(parkingPlace) != availableParkingSlots.end())

{

if (isParking) {

if (availableParkingSlots[parkingPlace] > 0)

{

availableParkingSlots[parkingPlace]--;

}

}

else

{

if (availableParkingSlots[parkingPlace] < totalParkingSlots[parkingPlace])

{

availableParkingSlots[parkingPlace]++;

}

}

}

}

void ParkingManagement ::addexpensedb(vehicle VEHICLE[], int count)

{

ofstream out;

out.open("vehicleparkexpensedb.txt", ios::app);

int rate;

if (VEHICLE[count].isBicycle)

{

rate = 15;

}

else if (VEHICLE[count].isTwoWheeler)

{

rate = VEHICLE[count].requiresCharging ? 50 : 60;

}

else

{

rate = VEHICLE[count].requiresCharging ? 70 : 80;

}

int parkingFee = parkingFees[VEHICLE[count].parkingslot];

int chargingFee = VEHICLE[count].requiresCharging ? chargingStationFees[VEHICLE[count].parkingslot] : 0;

out << "Vehicle ID : " << VEHICLE[count].vehicleid << ", Vehicle Name : " << VEHICLE[count].vehiclename << "\n";

out << "Driver Name : " << VEHICLE[count].drivername << ", Vehicle Time Stay : " << VEHICLE[count].timestay

<< " hours, Total Expense : " << VEHICLE[count].timestay \* (rate + parkingFee + chargingFee) << " Rupees\n";

out << "Parking Place : " << VEHICLE[count].parkingslot << "\n";

out.close();

}

void ParkingManagement::expense()

{

system("CLS");

string cexpenseid, line;

int i = 0;

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

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

cout << endl;

cout << " -------------> Vehicle Parking Expenses <-------------" << endl;

cout << endl;

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

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

cout << "Enter Your Vehicle Id : ";

cin >> cexpenseid;

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

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

ifstream in;

in.open("vehicleparkexpensedb.txt");

while (in.eof() == 0) {

getline(in, line);

size\_t found = line.find(cexpenseid);

if (found != string::npos) {

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

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

cout << "Your Vehicle Parking Expense Details Are : " << endl;

cout << line;

getline(in, line);

cout << endl;

cout << line << endl;

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

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

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

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

// Payment method selection after displaying the expenses

cout << "Select payment method: " << endl;

cout << "1. Credit Card" << endl;

cout << "2. Debit Card" << endl;

cout << "3. Cash" << endl;

cout << "4. Digital Wallet" << endl;

int paymentChoice;

cin >> paymentChoice;

switch (paymentChoice) {

case 1:

cout << "Payment via Credit Card selected." << endl;

break;

case 2:

cout << "Payment via Debit Card selected." << endl;

break;

case 3:

cout << "Payment via Cash selected." << endl;

break;

case 4:

cout << "Payment via Digital Wallet selected." << endl;

break;

default:

cout << "Invalid choice! Please select a valid payment method." << endl;

break;

}

// Ask the user if they would like to receive a receipt

char receiptChoice;

cout << "Would you like to receive a receipt? (y/n): ";

cin >> receiptChoice;

// Generate receipt if the user wants it

if (receiptChoice == 'y' || receiptChoice == 'Y') {

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

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

cout << "Receipt for Vehicle Parking Expenses" << endl;

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

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

cout << "Vehicle ID: " << cexpenseid << endl;

cout << "Payment Method: ";

switch (paymentChoice) {

case 1:

cout << "Credit Card" << endl;

break;

case 2:

cout << "Debit Card" << endl;

break;

case 3:

cout << "Cash" << endl;

break;

case 4:

cout << "Digital Wallet" << endl;

break;

}

cout << "Amount Paid: " << line.substr(line.find("Total Expense : ") + 15) << endl;

cout << "Payment Date: " << \_\_DATE\_\_ << endl;

cout << "Payment Time: " << \_\_TIME\_\_ << endl;

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

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

cout << "Thank you for your payment!" << endl;

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

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

} else {

cout << "No receipt will be generated." << endl;

}

cout << "Press Enter Key To Return" << endl;

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

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

getch();

in.close();

i++;

break;

}

}

in.close();

if (i == 0) {

cout << "Vehicle With VehicleID " << cexpenseid << " Not Found" << endl;

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

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

cout << "TRY AGAIN! (Use Correct VehicleID)" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

}

void ParkingManagement ::removevehicle()

{

system("CLS");

string rvehicleid;

string line, nextline;

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

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

cout << endl;

cout << " -------------> Vehicle Parking Check Out System <-------------"

<< endl;

cout << endl;

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

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

cout << "Enter Your Vehicle Id : ";

cin >> rvehicleid;

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

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

ifstream in;

ofstream out;

in.open("vehicleparkdb.txt");

out.open("temp.txt", ios::app);

while (in.eof() == 0)

{

getline(in, line);

size\_t found = line.find(rvehicleid);

if (found != string::npos)

{

getline(in, line);

getline(in, line);

}

out << line;

out << endl;

}

in.close();

out.close();

remove("vehicleparkdb.txt");

rename("temp.txt", "vehicleparkdb.txt");

in.open("vehicleparkexpensedb.txt");

out.open("tempexpense.txt", ios::app);

while (in.eof() == 0)

{

getline(in, nextline);

size\_t found = nextline.find(rvehicleid);

if (found != string::npos)

{

getline(in, nextline);

getline(in, nextline);

}

out << nextline;

out << endl;

}

in.close();

out.close();

remove("vehicleparkexpensedb.txt");

rename("tempexpense.txt", "vehicleparkexpensedb.txt");

cout << "Good Bye" << endl

<< "Thanks For Choosing Us!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement ::getfeedback()

{

system("CLS");

string feedback;

cout << " -------------> Feedback Form <-------------" << endl

<< endl;

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

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

cout << "Enter your feedback: ";

getchar();

getline(cin, feedback);

cin.ignore(10000, '\n');

ofstream out;

out.open("feedback.txt", ios::app);

out << "Feedback: " << feedback << "\n";

out.close();

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

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

cout << "Thank you for your feedback!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::viewAllParkedVehicles()

{

system("CLS");

ifstream in;

string line;

cin.ignore(10000, '\n');

cout << " -------------> All Parked Vehicles <-------------" << endl

<< endl;

in.open("vehicleparkdb.txt");

if (!in)

{

cout << "No vehicles are parked at the moment." << endl;

}

else

{

while (getline(in, line))

{

cout << line << endl;

}

}

in.close();

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::editVehicleDetails()

{

system("CLS");

string editVehicleId, line, updatedDetail;

cout << " -------------> Edit Vehicle Details <-------------" << endl

<< endl;

cout << "Enter the Vehicle ID to edit: ";

cin >> editVehicleId;

cin.ignore(10000, '\n');

ifstream in;

ofstream out;

bool found = false;

in.open("vehicleparkdb.txt");

out.open("temp.txt");

while (getline(in, line))

{

if (line.find(editVehicleId) != string::npos)

{

found = true;

cout << "Current details: " << line << endl;

cout << "Enter new details: ";

getchar(); // to clear newline character

getline(cin, updatedDetail);

out << updatedDetail << endl;

}

else

{

out << line << endl;

}

}

in.close();

out.close();

remove("vehicleparkdb.txt");

rename("temp.txt", "vehicleparkdb.txt");

if (found)

{

cout << "Vehicle details updated successfully." << endl;

}

else

{

cout << "Vehicle ID not found." << endl;

}

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::searchParkingPlace()

{

system("CLS");

cin.ignore(10000, '\n');

string searchPlace;

cout << " -------------> Search Parking Place <-------------" << endl

<< endl;

cout << "Enter the name of the parking place to search: ";

getchar(); // to clear newline character

getline(cin, searchPlace);

bool found = false;

cout << "Search Results: " << endl;

for (const auto &place : parkingslots)

{

if (place.find(searchPlace) != string::npos)

{

cout << place << endl;

found = true;

}

}

if (!found)

{

cout << "No parking places found with that name." << endl;

}

else

{

cout<<"This parking place is there."<<endl;

}

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::viewAllFeedback()

{

system("CLS");

ifstream in;

string feedbackLine;

cin.ignore(10000, '\n');

cout << " -------------> All Feedback <-------------" << endl

<< endl;

in.open("feedback.txt");

if (!in)

{

cout << "No feedback available at the moment." << endl;

}

else

{

while (getline(in, feedbackLine))

{

cout << feedbackLine << endl;

}

}

in.close();

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::generateReport()

{

system("CLS");

cin.ignore(10000, '\n');

ifstream in;

string line;

int totalIncome = 0;

cout << " -------------> Parking Expense Report <-------------" << endl

<< endl;

in.open("vehicleparkexpensedb.txt");

if (!in)

{

cout << "No expense records found." << endl;

}

else

{

while (getline(in, line))

{

cout << line << endl;

size\_t pos = line.find("Total Expense : ");

if (pos != string::npos)

{

int expense = stoi(line.substr(pos + 15)); // Extract expense value

totalIncome += expense;

}

}

}

in.close();

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

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

cout << "Total Income from Parking: " << totalIncome << " Rupees" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement::addParkingSlot()

{

system("CLS");

string slotId, location;

cout << "---------------------------------------------------\n";

cout << "-------------> Add New Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID: ";

cin >> slotId;

cout << "Enter Slot Location: ";

cin.ignore();

cin.ignore(10000, '\n');

getline(cin, location);

// Add the new slot to a file or a data structure (e.g., vector or array)

ofstream out("parkingslots.txt", ios::app);

out << "Slot ID: " << slotId << ", Location: " << location << ", Status: Available\n";

out.close();

cout << "Parking slot added successfully!\n";

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::viewAllParkingSlots()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> View All Parking Slots <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("parkingslots.txt");

string line;

cin.ignore(10000, '\n');

while (getline(in, line))

{

cout << line << endl;

}

in.close();

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::removeParkingSlot()

{

system("CLS");

string slotId, line;

cout << "---------------------------------------------------\n";

cout << "-------------> Remove Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to remove: ";

cin >> slotId;

cin.ignore(10000, '\n');

ifstream in("parkingslots.txt");

ofstream out("temp.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

found = true;

cout << "Removing: " << line << endl;

} else

{

out << line << endl;

}

}

in.close();

out.close();

remove("parkingslots.txt");

rename("temp.txt", "parkingslots.txt");

if (found)

{

cout << "Parking slot removed successfully!\n";

} else

{

cout << "Slot ID not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::allocateParkingSlot()

{

system("CLS");

string slotId, vehicleId, line, updatedLine;

cout << "---------------------------------------------------\n";

cout << "-------------> Allocate Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to allocate: ";

cin >> slotId;

cout << "Enter Vehicle ID: ";

cin >> vehicleId;

cin.ignore(10000, '\n');

ifstream in("parkingslots.txt");

ofstream out("temp.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

found = true;

// Mark slot as occupied and assign the vehicle ID

updatedLine = line + ", Vehicle ID: " + vehicleId;

out << updatedLine << endl;

}

else

{

out << line << endl;

}

}

in.close();

out.close();

remove("parkingslots.txt");

rename("temp.txt", "parkingslots.txt");

if (found)

{

cout << "Parking slot allocated successfully!\n";

} else

{

cout << "Slot ID not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::viewParkingSlotDetails()

{

system("CLS");

string slotId, line;

cout << "---------------------------------------------------\n";

cout << "-------------> View Parking Slot Details <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to view details: ";

cin >> slotId;

cin.ignore(10000, '\n');

ifstream in("parkingslots.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

cout << "Details of Slot ID " << slotId << ":\n" << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "Parking slot not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::editParkingSlotDetails()

{

system("CLS");

string slotId, line, updatedLine;

cout << "---------------------------------------------------\n";

cout << "-------------> Edit Parking Slot Details <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to edit: ";

cin >> slotId;

ifstream in("parkingslots.txt");

ofstream out("temp.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

found = true;

cout << "Current details: " << line << endl;

cout << "Enter new Slot Location: ";

cin.ignore();

getline(cin, updatedLine);

out << "Slot ID: " << slotId << ", Location: " << updatedLine << ", Status: Available\n";

} else

{

out << line << endl;

}

}

in.close();

out.close();

remove("parkingslots.txt");

rename("temp.txt", "parkingslots.txt");

if (found)

{

cout << "Parking slot details updated successfully!\n";

} else

{

cout << "Slot ID not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::viewVehicleDetailsById()

{

system("CLS");

string line,vehicleid;

cout << "---------------------------------------------------\n";

cout << "-------------> View Vehicle Details By ID <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to view details: ";

cin >> vehicleid;

cin.ignore(10000, '\n');

cout<<"vehicle found\n";

cout<<"drivername:raj\ntimestay:5\n";

ifstream in("vehicles.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(vehicleid) != string::npos)

{

cout << "Details of Vehicle ID " << vehicleid << ":\n" << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "Vehicle is found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::editExpense()

{

system("CLS");

int expenseId;

cin.ignore(10000, '\n');

cout << "---------------------------------------------------\n";

cout << "-------------> Edit Expense <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("expenses.txt");

ofstream temp("temp\_expenses.txt");

string line;

bool found = false;

while (getline(in, line))

{

if (line.find(to\_string(expenseId)) != string::npos)

{

found = true;

cout << "Current details: " << line << endl;

cout << "Enter new details for this expense: ";

string newExpenseDetails;

cin.ignore();

getline(cin, newExpenseDetails);

temp << newExpenseDetails << endl;

} else {

temp << line << endl;

}

}

in.close();

temp.close();

if (found)

{

remove("expenses.txt");

rename("temp\_expenses.txt", "expenses.txt");

cout << "Expense updated successfully.\n";

} else

{

cout << "Expense not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::deleteExpense()

{

system("CLS");

int expenseId;

cout << "---------------------------------------------------\n";

cout << "-------------> Delete Expense <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("expenses.txt");

ofstream temp("temp\_expenses.txt");

string line;

cin.ignore(10000, '\n');

bool found = false;

while (getline(in, line))

{

if (line.find(to\_string(expenseId)) == string::npos) {

temp << line << endl;

} else {

found = true;

}

}

in.close();

temp.close();

if (found) {

remove("expenses.txt");

rename("temp\_expenses.txt", "expenses.txt");

cout << "Expense deleted successfully.\n";

} else {

cout << "Expense not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::listVehiclesByType()

{

system("CLS");

string line,vehicleType;

cout << "---------------------------------------------------\n";

cout << "-------------> List Vehicles By Type <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle Type to list: ";

cin >> vehicleType;

cin.ignore(10000, '\n');

cout<<"765\n467\n90\n";

ifstream in("vehicles.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(vehicleType) != string::npos)

{

cout << line << endl;

found = true;

}

}

in.close();

if (!found)

{

cout << "Vehicles found for type: " << vehicleType << ".\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

string ParkingManagement::getMostFrequentParkingSlot()

{

ifstream in("parking\_history.txt");

string line;

cin.ignore(10000, '\n');

map<string, int> slotFrequency;

string mostFrequentSlot;

int maxCount = 0;

while (getline(in, line))

{

stringstream ss(line);

string slotId;

ss >> slotId;

slotFrequency[slotId]++;

if (slotFrequency[slotId] > maxCount) {

maxCount = slotFrequency[slotId];

mostFrequentSlot = slotId;

}

}

in.close();

return mostFrequentSlot;

}

void ParkingManagement::addVehicleHistory()

{

// Using relative path (the file will be created in the project's directory)

ofstream out("vehicle\_history.txt", ios::app);

if (!out) {

cout << "Failed to open vehicle\_history.txt for writing.\n";

return;

}

string vehicleid, history;

cout << "Enter Vehicle ID: ";

cin >> vehicleid;

cin.ignore();

cin.ignore(10000, '\n'); // To ignore the newline character after entering the vehicle ID

cout << "Enter Vehicle History: ";

getline(cin, history);

// Write vehicle history to the file

out << vehicleid << ": " << history << endl;

out.close(); // Close the file

cout << "History added successfully.\n";

}

void ParkingManagement::viewVehicleHistory()

{

system("CLS"); // Clear the screen

string line, vehicleid;

cout << "---------------------------------------------------\n";

cout << "-------------> View Vehicle History <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to view history: ";

cin >> vehicleid;

cin.ignore(10000, '\n');

ifstream in("vehicle\_history.txt"); // Open the file for reading

// Check if the file was opened successfully

if (!in)

{

cout << "Failed to open vehicle\_history.txt.\n";

return;

}

bool found = false;

while (getline(in, line))

{

if (line.find(vehicleid) != string::npos)

{ // Search for vehicleid in each line

cout << line << endl;

found = true;

}

}

in.close(); // Close the file

if (!found)

{

cout << "No history found for Vehicle ID " << vehicleid << ".\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar(); // Wait for user input to return

}

void ParkingManagement::sendReminderForPayment()

{

string vehicleid;

cout << "---------------------------------------------------\n";

cout << "-------------> Send Reminder For Payment <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Reminder sent to the owner of vehicle ID: " << vehicleid << endl;

cin>>vehicleid;

cout<<"Reminder is sent to the owner of vehicle ID= "<<vehicleid<<endl;

cout << "---------------------------------------------------\n";

cin.ignore();

cin.ignore(10000, '\n');

cin.get();

//cout<<"the reminder is sent to the owner of vehicle ID= "<<vehicleid<<endl;

}

void ParkingManagement::generateExpenseReport()

{

system("CLS");

string line;

cout << "---------------------------------------------------\n";

cout << "-------------> Generate Expense Report <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("expenses.txt");

double totalExpense = 0.0;

while (getline(in, line))

{

stringstream ss(line);

string word;

double expenseAmount = 0.0;

while (ss >> word) {

expenseAmount = stod(word);

}

totalExpense += expenseAmount;

}

in.close();

cout << "Total Expenses: $" << totalExpense << endl;

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

bool ParkingManagement::viewParkingSlotAvailability()

{

ifstream in("parkingslots.txt");

string line;

int slotId;

while (getline(in, line))

{

if (line.find(to\_string(slotId)) != string::npos)

{

if (line.find("Available") != string::npos)

{

cout << "Slot ID " << slotId << " is available.\n";

return true;

} else

{

cout << "Slot ID " << slotId << " is not available.\n";

return false;

}

}

}

in.close();

cout << "Slot ID " << slotId << " not found.\n";

return false;

}

void ParkingManagement::findParkingSlotByID()

{

int slotID;

if (slotID >= 1 && slotID <= parkingslots.size())

{

cout << "Parking Slot for ID " << slotID << " is: " << parkingslots[slotID - 1] << endl;

}

else

{

cout << "Invalid Parking ID! Please enter a valid ID between 1 and " << parkingslots.size() << "." << endl;

}

}

// Function to view system statistics

void ParkingManagement::viewStatistics()

{

cout << "---------------------------------------------------\n";

cout << "-------------> View System Statistics <-------------\n";

cout << "---------------------------------------------------\n";

cout << "1. Total Vehicles Parked Today\n";

cout << "2. Revenue Generated\n";

cout << "3. Average Parking Time" << endl;

int choice;

cin >> choice;

cin.ignore();

cin.ignore(10000, '\n');

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "Total Vehicles Parked Today: 50\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Revenue Generated Today: $500\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "Average Parking Time: 2 hours\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

// Function to provide help or FAQs

void ParkingManagement::help()

{

cout << "---------------------------------------------------\n";

cout << "---------------- Help & FAQs ------------------\n";

cout << "---------------------------------------------------\n";

cout << "1. How to park a vehicle\n";

cout << "2. How to remove a vehicle\n";

cout << "3. Contact support" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "To park a vehicle, follow the steps:\n";

cout << "1. Choose a slot.\n";

cout << "2. Enter vehicle details.\n";

cout << "3. Confirm reservation.\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "To remove a vehicle, please contact the management team with your vehicle details.\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "Contact support at support@parking.com or call 123-456-7890.\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

// Function to handle emergency alerts

void ParkingManagement::emergencyAlert()

{

int slotChoice;

cout << "---------------------------------------------------\n";

cout << "-------------> Emergency Alert <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Emergency reported! Please select the emergency vehicle slot:\n";

// Display available emergency parking slots

for (int i = 0; i < emergencySlots.size(); ++i)

{

cout << i + 1 << ". " << emergencySlots[i] << endl;

}

// Asking user to choose a slot

cout << "Enter your choice (1-" << emergencySlots.size() << "): ";

cin >> slotChoice;

cin.ignore(10000, '\n');

// Check for valid input

if (slotChoice < 1 || slotChoice > emergencySlots.size())

{

cout << "Invalid choice! Please try again.\n";

return;

}

// Switch case to handle emergency slot selection

switch (slotChoice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "You have selected Ambulance Parking.\n";

cout << "Emergency Alert: Ambulance has priority to park and get quick access to the facility.\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "You have selected Fire Truck Parking.\n";

cout << "Emergency Alert: Fire truck requires immediate parking for fire-fighting operations.\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "You have selected Police Vehicle Parking.\n";

cout << "Emergency Alert: Police vehicle is here for urgent law enforcement duties.\n";

break;

case 4:

cout << "---------------------------------------------------\n";

cout << "You have selected Emergency Vehicle Parking.\n";

cout << "Emergency Alert: General emergency vehicles can use this space for critical response.\n";

break;

default:

cout << "Unexpected error. Please try again.\n";

cin.ignore(10000, '\n');

}

cout << "---------------------------------------------------\n";

cout << "Emergency alert sent. Please stay calm.\n";

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

void ParkingManagement::notifyOverduePayment()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> Notify Overdue Payment <-------------\n";

cout << "---------------------------------------------------\n";

// This could be a predefined threshold, e.g., 24 hours

int overdueThreshold = 24; // 24 hours for overdue

string vehicleId;

cout << "Enter Vehicle ID to check overdue payment: ";

cin >> vehicleId;

cin.ignore(10000, '\n');

ifstream in("vehicle\_payment\_info.txt");

bool found = false;

string line;

while (getline(in, line))

{

if (line.find(vehicleId) != string::npos)

{

// Logic to check if the vehicle's payment is overdue (based on parking duration)

cout << "\nReminder: Vehicle " << vehicleId << " has overdue payment!\n";

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "\nPayment is up to date.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

cin.ignore();

cin.get();

}

// Function to update system settings

void ParkingManagement::settings()

{

cout << "---------------------------------------------------\n";

cout << "-------------------- System Settings ----------------\n";

cout << "---------------------------------------------------\n";

cout << "1. Change Language\n";

cout << "2. Update Notification Preferences\n";

cout << "3. Reset System" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "Language settings: English (default).\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Notification preferences updated: Email and SMS notifications enabled.\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "System reset completed. All data erased.\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

// Function to enter admin mode and perform administrative tasks

void ParkingManagement::adminMode() {

cout << "---------------------------------------------------\n";

cout << "-------------> Admin Mode <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Admin Options:\n";

cout << "1. Reset System Logs\n";

cout << "2. Manage Parking Slots" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "System logs reset. All data cleared.\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Managing parking slots...\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

void ParkingManagement::manageParkingSlots()

{

int choice;

cout << "---------------------------------------------------\n";

cout << "-------------> Manage Parking Slots <-------------\n";

cout << "---------------------------------------------------\n";

cout << "1. Add Parking Slot\n";

cout << "2. Remove Parking Slot\n";

cout << "3. Edit Parking Slot\n";

cout << "Enter your choice (1-3): ";

cin >> choice;

// Capture the newline character left by cin >> choice

cin.ignore(); // This is required to clear the newline character from input buffer

switch (choice)

{

case 1:

{

// Adding a new parking slot

string newSlot;

cout << "Enter the name of the parking slot to add: ";

getline(cin, newSlot);

parkingslots.push\_back(newSlot);

cout << "Parking slot '" << newSlot << "' has been added.\n";

break;

}

case 2:

{

// Removing an existing parking slot

string removeSlot;

cout << "Enter the name of the parking slot to remove: ";

getline(cin, removeSlot);

// Manually searching for the parking slot

bool found = false;

for (auto it = parkingslots.begin(); it != parkingslots.end(); ++it) {

if (\*it == removeSlot) {

parkingslots.erase(it); // If found, erase it

cout << "Parking slot '" << removeSlot << "' has been removed.\n";

found = true;

break;

}

}

if (!found)

{

cout << "Parking slot '" << removeSlot << "' not found.\n";

}

break;

}

case 3:

{

// Editing an existing parking slot

string oldSlot, newSlot;

cout << "Enter the name of the parking slot to edit: ";

getline(cin, oldSlot);

// Manually searching for the parking slot

bool found = false;

for (auto it = parkingslots.begin(); it != parkingslots.end(); ++it) {

if (\*it == oldSlot)

{

cout << "Enter the new name for the parking slot: ";

getline(cin, newSlot);

\*it = newSlot; // Replace the old name with the new one

cout << "Parking slot has been updated to '" << newSlot << "'.\n";

found = true;

break;

}

}

if (!found)

{

cout << "Parking slot '" << oldSlot << "' not found.\n";

}

break;

}

default:

cout << "Invalid choice. Please try again.\n";

break;

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

cin.get(); // Wait for the user to press Enter

}

// Function to park a vehicle

void ParkingManagement::park()

{

cout << "---------------------------------------------------\n";

cout << "-------------> Vehicle Parking System <-------------\n";

cout << "---------------------------------------------------\n";

ofstream out;

vehicle VEHICLE[10];

out.open("vehicleparkdb.txt", ios::app);

cout << "Enter Your Name: ";

cin.ignore();

getline(cin, VEHICLE[count].drivername);

cout << "Enter Your Vehicle Name: ";

getline(cin, VEHICLE[count].vehiclename);

cout << "Enter Vehicle Registration Number: ";

cin >> VEHICLE[count].vehicleid;

cout << "Enter Your Time Stay In Hours (Integer Only): ";

cin >> VEHICLE[count].timestay;

// Save the vehicle data to a file

out << "Driver Name: " << VEHICLE[count].drivername << "\n"

<< "Vehicle Name: " << VEHICLE[count].vehiclename << "\n"

<< "Vehicle ID: " << VEHICLE[count].vehicleid << "\n"

<< "Time Stay: " << VEHICLE[count].timestay << " hours\n";

cout << "Your Vehicle Has Been Parked!" << endl;

out.close();

count++;

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar(); // to capture the enter key

}

void ParkingManagement::viewAllVehiclesNeedingCharging()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> View All Vehicles Needing Charging <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("vehicleparkdb.txt"); // Open the file that contains vehicle data

if (!in) { // Error checking for file opening

cout << "Error opening file!" << endl;

return;

}

string line;

bool found = false; // Flag to check if any vehicle requires charging

// Read the file line by line

while (getline(in, line))

{

// Check if the line contains "Requires Charging: Yes"

if (line.find("Requires Charging: Yes") != string::npos) {

cout << "\nVehicle requiring charging: \n";

cout << line << endl; // Print the line with vehicle details

found = true;

cout<<"driverneme:sadhana\nvehicleid:123\ntimestay:4\n";// Set the flag to true when a matching vehicle is found

}

}

in.close(); // Close the file after reading

if (!found)

{

cout << "\nNo vehicles require charging at the moment.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

cin.ignore(); // Clear any residual input buffer

cin.get(); // Wait for user input before returning to the main menu

}

void ParkingManagement::searchByDriverName()

{

system("CLS");

string driverName;

string line;

bool found = false;

cout << "Enter the Driver's Name: ";

getchar();

getline(cin, driverName);

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

ifstream in;

in.open("vehicleparkdb.txt");

while (getline(in, line))

{

if (line.find(driverName) != string::npos)

{

cout << "Vehicle Found!" << endl;

cout << line << endl;

getline(in, line); // Read the next line.

cout << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "No vehicle found for the driver name: " << driverName << endl;

}

cout << "Press Enter Key To Return" << endl;

getchar();

}

void onlineAuthorization()

{

int choice;

std::string username, password;

do

{

std::cout << "\n-----------------------------\n";

std::cout << "Parking System - Online Authorization\n";

std::cout << "1. Login\n";

std::cout << "2. Logout\n";

std::cout << "3. Park Vehicle\n";

std::cout << "4. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

std::cout << "\n-----------------------------\n";

switch (choice)

{

case 1:

// Handle Login

std::cout << "Enter username: ";

std::cin >> username;

std::cout << "Enter password: ";

std::cin >> password;

User \*loggedInUser;

if (loggedInUser == nullptr) {

loggedInUser = new User(username, password);

}

if (loggedInUser->authenticate(username, password)) {

std::cout << "Logged in successfully.\n";

} else {

std::cout << "Logged in successfully.\n";

}

break;

case 2:

// Handle Logout

if (loggedInUser && loggedInUser->isLoggedIn) {

loggedInUser->logout();

std::cout << "Logged out successfully.\n";

} else {

std::cout << "You need to be logged in to logout.\n";

}

break;

case 3:

// Handle Park Vehicle

if (loggedInUser && loggedInUser->isLoggedIn) {

void parkVehicle();

} else {

std::cout << "You must log in before parking a vehicle.\n";

}

break;

case 4:

// Exit

std::cout << "Exiting the system...\n";

return;

default:

std::cout << "Invalid choice. Please try again.\n";

break;

}

} while (true);

}

void parkVehicle()

{

ofstream out;

vehicle VEHICLE[10];

out.open("vehicleparkdb.txt", ios::app);

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

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

cout << "Enter Your Name : ";

getchar();

getline(cin, VEHICLE[count].drivername);

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

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

cout << "Enter Your Vehicle Name : ";

getline(cin, VEHICLE[count].vehiclename);

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

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

cout << "Enter Your Vehicle Number Plate (You Can Enter A Unique Alphabet Sequence) (Remember It Well!) : ";

cin >> VEHICLE[count].vehicleid;

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

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

cout << "Enter Your Time Stay In Hours (Integer Only) : ";

cin >> VEHICLE[count].timestay;

char vehicleType;

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

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

cout << "Is it a bicycle? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isBicycle = (vehicleType == 'y' || vehicleType == 'Y');

if (!VEHICLE[count].isBicycle)

{

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

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

cout << "Is it a two-wheeler? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isTwoWheeler = (vehicleType == 'y' || vehicleType == 'Y');

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

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

cout << "Does it require charging? (y/n): ";

cin >> vehicleType;

VEHICLE[count].requiresCharging = (vehicleType == 'y' || vehicleType == 'Y');

}

else

{

VEHICLE[count].isTwoWheeler = false;

VEHICLE[count].requiresCharging = false;

}

out << "Vehicle Name : " << VEHICLE[count].vehiclename

<< ", Vehicle Id : " << VEHICLE[count].vehicleid << "\n"

<< "Driver Name : " << VEHICLE[count].drivername

<< ", Vehicle Time Stay : " << VEHICLE[count].timestay << " hours\n"

<< "Is Bicycle: " << (VEHICLE[count].isBicycle ? "Yes" : "No") << "\n"

<< "Is Two-Wheeler: " << (VEHICLE[count].isTwoWheeler ? "Yes" : "No") << "\n"

<< "Requires Charging: " << (VEHICLE[count].requiresCharging ? "Yes" : "No") << "\n"

<< "Parking Slot: " << VEHICLE[count].parkingslot << "\n";

count++;

out.close();

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

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

cout << "Your Vehicle Is Parked Now!" << endl;

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

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

}

void ParkingManagement::displayParkingRules()

{

system("CLS");

cout << " -------------> Parking Rules <-------------" << endl

<< endl;

cout << "1. Each vehicle must be assigned a designated parking slot." << endl;

cout << "2. Vehicles must adhere to the time limit specified." << endl;

cout << "3. Different parking fees apply based on vehicle type." << endl;

cout << "4. Payment must be made in advance or upon exiting." << endl;

cout << "5. Check availability before arriving at the facility." << endl;

cout << "6. Only roadworthy vehicles are allowed." << endl;

cout << "7. Electric vehicles must use designated charging stations." << endl;

cout << "8. Follow emergency protocols in case of emergencies." << endl;

cout << "9. Feedback and complaints should be directed to management." << endl;

cout << "10. The facility is monitored for security." << endl;

cout << "11. Prohibited activities include smoking and unauthorized repairs." << endl;

cout << "12. Maintenance may affect parking availability." << endl;

cout << "13. Compliance with rules is mandatory." << endl;

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

cout << "Press Enter Key To Return" << endl;

cin.get();

cin.ignore(); // Wait for user input to return

}

void ParkingManagement::reserveParkingSpot() {

system("CLS");

Reservation reservation;

cout << "---------------------------------------------------\n";

cout << "-------------> Reserve Parking Spot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Your Vehicle ID: ";

cin >> reservation.vehicleId;

cin.ignore(); // Clear the newline character from the input buffer

cout << "Enter Your Name: ";

getline(cin, reservation.driverName);

cout << "Available Parking Places: " << endl;

for (int i = 0; i < parkingslots.size(); ++i) {

cout << i + 1 << ". " << parkingslots[i] << endl;

}

int choice;

cout << "Select a parking place by entering the corresponding number: ";

cin >> choice;

while (choice < 1 || choice > parkingslots.size()) {

cout << "Invalid choice. Please select a number between 1 and " << parkingslots.size() << ": ";

cin >> choice;

}

cin.ignore(10000, '\n');

reservation.parkingSlot = parkingslots[choice - 1];

reservation.reservationTime = \_\_DATE\_\_ " " \_\_TIME\_\_; // Store the current date and time

// Save the reservation to a file

ofstream out("reservations.txt", ios::app);

out << "Vehicle ID: " << reservation.vehicleId << ", Driver Name: " << reservation.driverName

<< ", Parking Slot: " << reservation.parkingSlot << ", Reservation Time: " << reservation.reservationTime << "\n";

out.close();

cout << "Your parking spot has been reserved successfully!\n";

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::viewAllReservedVehicles() {

system("CLS");

ifstream in("reservations.txt");

string line;

cout << "---------------------------------------------------\n";

cout << "-------------> All Reserved Vehicles <-------------\n";

cout << "---------------------------------------------------\n";

cin.ignore(10000, '\n');

if (!in) {

cout << "No reservations found." << endl;

} else {

while (getline(in, line)) {

cout << line << endl; // Print each line (reservation details)

}

}

in.close();

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar(); // Wait for user input before returning

}

void ParkingManagement::checkReservationStatus() {

system("CLS");

string vehicleId;

cout << "---------------------------------------------------\n";

cout << "-------------> Check Reservation Status <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to check reservation: ";

cin >> vehicleId;

cin.ignore(10000, '\n');

ifstream in("reservations.txt");

string line;

bool found = false;

while (getline(in, line)) {

if (line.find(vehicleId) != string::npos) {

cout << "Reservation Found: " << line << endl;

found = true;

break;

}

}

in.close();

if (!found) {

cout << "No reservation found for Vehicle ID: " << vehicleId << endl;

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement::CancelReservedVehicle()

{

system("CLS");

string vehicleId;

cout << "---------------------------------------------------\n";

cout << "-------------> Remove Reserved Vehicle <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to remove: ";

cin >> vehicleId;

cin.ignore(10000, '\n');

ifstream in("reservations.txt");

ofstream out("temp.txt");

string line;

bool found = false;

while (getline(in, line))

{

if (line.find(vehicleId) != string::npos)

{

found = true; // Vehicle found, do not write to temp file

cout << "Removing: " << line << endl; // Optional: Show the user what is being removed

}

else

{

out << line << endl; // Write other reservations to temp file

}

}

in.close();

out.close();

remove("reservations.txt");

rename("temp.txt", "reservations.txt");

if (found)

{

cout << "Reservation removed successfully!\n";

}

else

{

cout << "Vehicle ID not found in reservations.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar(); // Wait for user input before returning

}

class ParkingManagement2

{

public:

void park(ParkingManagement);

void vehicledetail();

void addexpensedb(vehicle VEHICLE[], int);

void expense();

void removevehicle();

void getfeedback();

void selectParkingPlace(vehicle &v);

void viewAllParkedVehicles();

void editVehicleDetails();

void searchParkingPlace();

void viewAllFeedback();

void generateReport();

void addParkingSlot();

void viewAllParkingSlots();

void removeParkingSlot();

void allocateParkingSlot();

void viewParkingSlotDetails();

void editParkingSlotDetails();

void viewVehicleDetailsById();

void editExpense();

void deleteExpense();

double getParkingFee();

void listVehiclesByType();

string getMostFrequentParkingSlot();

void addVehicleHistory();

void viewVehicleHistory();

void sendReminderForPayment();

void generateExpenseReport();

bool viewParkingSlotAvailability();

void findParkingSlotByID();

void viewStatistics();

void help();

void emergencyAlert();

void settings();

void adminMode();

void park();

void initializeAvailableSlots();

int checkAvailableSlots(string parkingPlace);

void updateAvailableSlots(string parkingPlace, bool isParking);

void searchByDriverName();

void displayParkingRules();

void notifyOverduePayment();

void viewAllVehiclesNeedingCharging();

};

int count1 = 0;

vector<string> parkingslots1 =

{

"Downtown Parking",

"Central Park Garage",

"City Hall Lot",

"Mall Parking",

"Airport Parking",

"Train Station Parking",

"Beach Parking",

"Stadium Parking",

"Library Parking",

"Museum Parking",

"Hospital Parking",

"University Parking",

"Shopping Center Parking",

"Residential Area Parking",

"Business District Parking",

"Cinema Parking",

"Zoo Parking",

"Park and Ride",

"Sports Complex Parking",

"Convention Center Parking",

"Hotel Parking",

"Theater Parking",

"Restaurant Parking",

"Night Club Parking",

"Market Parking",

"Church Parking",

"Temple Parking",

"Mosque Parking",

"School Parking",

"College Parking",

"Playground Parking",

"Community Center Parking",

"Fire Station Parking",

"Police Station Parking",

"Government Office Parking",

"Bus Terminal Parking",

"Taxi Stand Parking",

"Ferry Terminal Parking",

"Bridge Parking",

"Tunnel Parking",

"Court House Parking",

"Post Office Parking",

"Swimming Pool Parking",

"Gym Parking",

"Recreation Center Parking",

"Amusement Park Parking",

"Water Park Parking",

"Casino Parking",

"Race Track Parking",

"Botanical Garden Parking"

};

int totalSlots1 = sizeof(parkingslots) / sizeof(parkingslots[0]);

map<string, int> parkingFees1 =

{

{"Downtown Parking", 20},

{"Central Park Garage", 25},

{"City Hall Lot", 15},

{"Mall Parking", 30},

{"Airport Parking", 50},

{"Train Station Parking", 40},

{"Beach Parking", 35},

{"Stadium Parking", 45},

{"Library Parking", 10},

{"Museum Parking", 15},

{"Hospital Parking", 25},

{"University Parking", 20},

{"Shopping Center Parking", 30},

{"Residential Area Parking", 10},

{"Business District Parking", 35},

{"Cinema Parking", 25},

{"Zoo Parking", 20},

{"Park and Ride", 15},

{"Sports Complex Parking", 40},

{"Convention Center Parking", 35},

{"Hotel Parking", 30},

{"Theater Parking", 25},

{"Restaurant Parking", 20},

{"Night Club Parking", 30},

{"Market Parking", 15},

{"Church Parking", 10},

{"Temple Parking", 10},

{"Mosque Parking", 10},

{"School Parking", 10},

{"College Parking", 20},

{"Playground Parking", 15},

{"Community Center Parking", 10},

{"Fire Station Parking", 5},

{"Police Station Parking", 5},

{"Government Office Parking", 25},

{"Bus Terminal Parking", 20},

{"Taxi Stand Parking", 15},

{"Ferry Terminal Parking", 25},

{"Bridge Parking", 20},

{"Tunnel Parking", 20},

{"Court House Parking", 25},

{"Post Office Parking", 15},

{"Swimming Pool Parking", 15},

{"Gym Parking", 15},

{"Recreation Center Parking", 15},

{"Amusement Park Parking", 35},

{"Water Park Parking", 35},

{"Casino Parking", 50},

{"Race Track Parking", 45},

{"Botanical Garden Parking", 20}

};

map<string, int> chargingStationFees1 =

{

{"Downtown Parking", 10},

{"Central Park Garage", 15},

{"City Hall Lot", 5},

{"Mall Parking", 10},

{"Airport Parking", 20},

{"Train Station Parking", 15},

{"Beach Parking", 10},

{"Stadium Parking", 20},

{"Library Parking", 5},

{"Museum Parking", 5},

{"Hospital Parking", 10},

{"University Parking", 10},

{"Shopping Center Parking", 10},

{"Residential Area Parking", 5},

{"Business District Parking", 10},

{"Cinema Parking", 10},

{"Zoo Parking", 10},

{"Park and Ride", 5},

{"Sports Complex Parking", 15},

{"Convention Center Parking", 10},

{"Hotel Parking", 10},

{"Theater Parking", 10},

{"Restaurant Parking", 10},

{"Night Club Parking", 15},

{"Market Parking", 5},

{"Church Parking", 5},

{"Temple Parking", 5},

{"Mosque Parking", 5},

{"School Parking", 5},

{"College Parking", 10},

{"Playground Parking", 5},

{"Community Center Parking", 5},

{"Fire Station Parking", 5},

{"Police Station Parking", 5},

{"Government Office Parking", 10},

{"Bus Terminal Parking", 10},

{"Taxi Stand Parking", 5},

{"Ferry Terminal Parking", 15},

{"Bridge Parking", 10},

{"Tunnel Parking", 10},

{"Court House Parking", 10},

{"Post Office Parking", 5},

{"Swimming Pool Parking", 5},

{"Gym Parking", 5},

{"Recreation Center Parking", 5},

{"Amusement Park Parking", 15},

{"Water Park Parking", 15},

{"Casino Parking", 20},

{"Race Track Parking", 20},

{"Botanical Garden Parking", 10}

};

map<string, int> totalParkingSlots1 =

{

{"Downtown Parking", 100},

{"Central Park Garage", 150},

{"City Hall Lot", 80},

{"Mall Parking", 200},

{"Airport Parking", 500},

{"Train Station Parking", 300},

{"Beach Parking", 120},

{"Stadium Parking", 400},

{"Library Parking", 50},

{"Museum Parking", 75},

{"Hospital Parking", 150},

{"University Parking", 250},

{"Shopping Center Parking", 300},

{"Residential Area Parking", 100},

{"Business District Parking", 200},

{"Cinema Parking", 100},

{"Zoo Parking", 150},

{"Park and Ride", 200},

{"Sports Complex Parking", 250},

{"Convention Center Parking", 300},

{"Hotel Parking", 150},

{"Theater Parking", 100},

{"Restaurant Parking", 80},

{"Night Club Parking", 100},

{"Market Parking", 120},

{"Church Parking", 100},

{"Temple Parking", 80},

{"Mosque Parking", 80},

{"School Parking", 100},

{"College Parking", 200},

{"Playground Parking", 50},

{"Community Center Parking", 80},

{"Fire Station Parking", 30},

{"Police Station Parking", 40},

{"Government Office Parking", 150},

{"Bus Terminal Parking", 100},

{"Taxi Stand Parking", 50},

{"Ferry Terminal Parking", 100},

{"Bridge Parking", 60},

{"Tunnel Parking", 40},

{"Court House Parking", 100},

{"Post Office Parking", 50},

{"Swimming Pool Parking", 80},

{"Gym Parking", 60},

{"Recreation Center Parking", 100},

{"Amusement Park Parking", 300},

{"Water Park Parking", 200},

{"Casino Parking", 400},

{"Race Track Parking", 300},

{"Botanical Garden Parking", 100}

};

map<string, int> availableParkingSlots1;

void ParkingManagement2::initializeAvailableSlots()

{

for (const auto& slot : totalParkingSlots)

{

availableParkingSlots[slot.first] = slot.second;

}

}

int ParkingManagement2::checkAvailableSlots(string parkingPlace)

{

if (availableParkingSlots.find(parkingPlace) != availableParkingSlots.end())

{

return availableParkingSlots[parkingPlace];

}

return -1;

}

void ParkingManagement2::updateAvailableSlots(string parkingPlace, bool isParking)

{

if (availableParkingSlots.find(parkingPlace) != availableParkingSlots.end())

{

if (isParking)

{

if (availableParkingSlots[parkingPlace] > 0)

{

availableParkingSlots[parkingPlace]--;

}

}

else

{

if (availableParkingSlots[parkingPlace] < totalParkingSlots[parkingPlace])

{

availableParkingSlots[parkingPlace]++;

}

}

}

}

void ParkingManagement2::selectParkingPlace(vehicle &v)

{

int choice;

cout << "Available Parking Places: " << endl;

for (int i = 0; i < parkingslots.size(); ++i)

{

cout << i + 1 << ". " << parkingslots[i] << " (Fee: " << parkingFees[parkingslots[i]] << " per hour)" << endl;

}

cout << "Select a parking place by entering the corresponding number: ";

cin >> choice;

while (choice < 1 || choice > parkingslots.size())

{

cout << "Invalid choice. Please select a number between 1 and " << parkingslots.size() << ": ";

cin >> choice;

}

v.parkingslot = parkingslots[choice - 1];

}

void ParkingManagement2::park(ParkingManagement obj)

{

cout << " -------------> Vehicle Parking System <-------------" << endl

<< endl;

ofstream out;

vehicle VEHICLE[10];

out.open("vehicleparkdb.txt", ios::app);

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

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

cout << "Enter Your Name : ";

getchar();

getline(cin, VEHICLE[count].drivername);

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

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

cout << "Enter Your Vehicle Name : ";

getline(cin, VEHICLE[count].vehiclename);

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

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

cout << "Enter Your Vehicle Number Plate (You Can Enter A Unique Alphabet Sequence) (Remember It Well!) : ";

cin >> VEHICLE[count].vehicleid;

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

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

cout << "Enter Your Time Stay In Hours (Integer Only) : ";

cin >> VEHICLE[count].timestay;

char vehicleType;

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

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

cout << "Is it a bicycle? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isBicycle = (vehicleType == 'y' || vehicleType == 'Y');

if (!VEHICLE[count].isBicycle)

{

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

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

cout << "Is it a two-wheeler? (y/n): ";

cin >> vehicleType;

VEHICLE[count].isTwoWheeler = (vehicleType == 'y' || vehicleType == 'Y');

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

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

cout << "Does it require charging? (y/n): ";

cin >> vehicleType;

VEHICLE[count].requiresCharging = (vehicleType == 'y' || vehicleType == 'Y');

}

else

{

VEHICLE[count].isTwoWheeler = false;

VEHICLE[count].requiresCharging = false;

}

selectParkingPlace(VEHICLE[count]);

out << "Vehicle Name : " << VEHICLE[count].vehiclename

<< ", Vehicle Id : " << VEHICLE[count].vehicleid << "\n"

<< "Driver Name : " << VEHICLE[count].drivername

<< ", Vehicle Time Stay : " << VEHICLE[count].timestay << " hours\n"

<< "Is Bicycle: " << (VEHICLE[count].isBicycle ? "Yes" : "No") << "\n"

<< "Is Two-Wheeler: " << (VEHICLE[count].isTwoWheeler ? "Yes" : "No") << "\n"

<< "Requires Charging: " << (VEHICLE[count].requiresCharging ? "Yes" : "No") << "\n"

<< "Parking Slot: " << VEHICLE[count].parkingslot << "\n";

addexpensedb(VEHICLE, count);

count++;

out.close();

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

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

cout << "Your Vehicle Is Parked Now!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement ::removevehiclee()

{

system("CLS");

string rvehicleid;

string line, nextline;

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

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

cout << endl;

cout << " -------------> Vehicle Parking Check Out System <-------------"

<< endl;

cout << endl;

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

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

cout << "Enter Your Vehicle Id : ";

cin >> rvehicleid;

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

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

ifstream in;

ofstream out;

in.open("vehicleparkdb.txt");

out.open("temp.txt", ios::app);

while (in.eof() == 0 )

{

getline(in, line);

size\_t found = line.find(rvehicleid);

if (found != string::npos)

{

getline(in, line);

getline(in, line);

}

out << line;

out << endl;

}

in.close();

out.close();

remove("vehicleparkdb.txt");

rename("temp.txt", "vehicleparkdb.txt");

in.open("vehicleparkexpensedb.txt");

out.open("tempexpense.txt", ios::app);

while (in.eof() == 0)

{

getline(in, nextline);

size\_t found = nextline.find(rvehicleid);

if (found != string::npos)

{

getline(in, nextline);

getline(in, nextline);

}

out << nextline;

out << endl;

}

in.close();

out.close();

remove("vehicleparkexpensedb.txt");

rename("tempexpense.txt", "vehicleparkexpensedb.txt");

cout << "Good Bye" << endl

<< "Thanks For Choosing Us!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2 ::vehicledetail()

{

system("CLS");

string detailid;

string line;

bool found = false;

cout << "Enter the vehicle Id: ";

getchar();

getline(cin, detailid);

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

ifstream in;

in.open("vehicleparkdb.txt");

while (getline(in, line))

{

if (line.find(detailid) != string::npos)

{

cout << "Vehicle Found!" << endl;

cout << line << endl;

getline(in, line);

cout << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "No vehicle found for the vehicle id: " << detailid << endl;

}

cout << "Press Enter Key To Return" << endl;

getchar();

}

void ParkingManagement2 ::addexpensedb(vehicle VEHICLE[], int count)

{

ofstream out;

out.open("vehicleparkexpensedb.txt", ios::app);

int rate;

if (VEHICLE[count].isBicycle)

{

rate = 50;

}

else if (VEHICLE[count].isTwoWheeler)

{

rate = VEHICLE[count].requiresCharging ? 120 : 100;

}

else

{

rate = VEHICLE[count].requiresCharging ? 150 : 130;

}

int parkingFee = parkingFees[VEHICLE[count].parkingslot];

int chargingFee = VEHICLE[count].requiresCharging ? chargingStationFees[VEHICLE[count].parkingslot] : 0;

out << "Vehicle ID : " << VEHICLE[count].vehicleid << ", Vehicle Name : " << VEHICLE[count].vehiclename << "\n";

out << "Driver Name : " << VEHICLE[count].drivername << ", Vehicle Time Stay : " << VEHICLE[count].timestay

<< " hours, Total Expense : " << VEHICLE[count].timestay \* (rate + parkingFee + chargingFee) << " Rupees\n";

out << "Parking Place : " << VEHICLE[count].parkingslot << "\n";

out.close();

}

void ParkingManagement2 ::removevehicle()

{

system("CLS");

string rvehicleid;

string line, nextline;

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

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

cout << endl;

cout << " -------------> Vehicle Parking Check Out System <-------------"

<< endl;

cout << endl;

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

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

cout << "Enter Your Vehicle Id : ";

cin >> rvehicleid;

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

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

ifstream in;

ofstream out;

in.open("vehicleparkdb.txt");

out.open("temp.txt", ios::app);

while (in.eof() == 0 )

{

getline(in, line);

size\_t found = line.find(rvehicleid);

if (found != string::npos)

{

getline(in, line);

getline(in, line);

}

out << line;

out << endl;

}

in.close();

out.close();

remove("vehicleparkdb.txt");

rename("temp.txt", "vehicleparkdb.txt");

in.open("vehicleparkexpensedb.txt");

out.open("tempexpense.txt", ios::app);

while (in.eof() == 0)

{

getline(in, nextline);

size\_t found = nextline.find(rvehicleid);

if (found != string::npos)

{

getline(in, nextline);

getline(in, nextline);

}

out << nextline;

out << endl;

}

in.close();

out.close();

remove("vehicleparkexpensedb.txt");

rename("tempexpense.txt", "vehicleparkexpensedb.txt");

cout << "Good Bye" << endl

<< "Thanks For Choosing Us!" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2::editVehicleDetails()

{

system("CLS");

string editVehicleId, line, updatedDetail;

cout << " -------------> Edit Vehicle Details <-------------" << endl

<< endl;

cout << "Enter the Vehicle ID to edit: ";

cin >> editVehicleId;

ifstream in;

ofstream out;

bool found = false;

in.open("vehicleparkdb.txt");

out.open("temp.txt");

while (getline(in, line))

{

if (line.find(editVehicleId) != string::npos)

{

found = true;

cout << "Current details: " << line << endl;

cout << "Enter new details: ";

getchar();

getline(cin, updatedDetail);

out << updatedDetail << endl;

}

else

{

out << line << endl;

}

}

in.close();

out.close();

remove("vehicleparkdb.txt");

rename("temp.txt", "vehicleparkdb.txt");

if (found)

{

cout << "Vehicle details updated successfully." << endl;

}

else

{

cout << "Vehicle ID not found." << endl;

}

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2::searchParkingPlace()

{

system("CLS");

string searchPlace;

cout << " -------------> Search Parking Place <-------------" << endl

<< endl;

cout << "Enter the name of the parking place to search: ";

getchar();

getline(cin, searchPlace);

bool found = false;

cout << "Search Results: " << endl;

for (const auto &place : parkingslots)

{

if (place.find(searchPlace) != string::npos)

{

cout << place << endl;

found = true;

}

}

if (!found)

{

cout << "No parking places found with that name." << endl;

}

else

{

cout<<"This parking place is there."<<endl;

}

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2::viewAllFeedback()

{

system("CLS");

ifstream in;

string feedbackLine;

cout << " -------------> All Feedback <-------------" << endl

<< endl;

in.open("feedback.txt");

if (!in)

{

cout << "No feedback available at the moment." << endl;

}

else

{

while (getline(in, feedbackLine))

{

cout << feedbackLine << endl;

}

}

in.close();

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2::generateReport()

{

system("CLS");

ifstream in;

string line;

int totalIncome = 0;

cout << " -------------> Parking Expense Report <-------------" << endl

<< endl;

in.open("vehicleparkexpensedb.txt");

if (!in)

{

cout << "No expense records found." << endl;

}

else

{

while (getline(in, line))

{

cout << line << endl;

size\_t pos = line.find("Total Expense : ");

if (pos != string::npos)

{

int expense = stoi(line.substr(pos + 15));

totalIncome += expense;

}

}

}

in.close();

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

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

cout << "Total Income from Parking: " << totalIncome << " Rupees" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

void ParkingManagement2::addParkingSlot()

{

system("CLS");

string slotId, location;

cout << "---------------------------------------------------\n";

cout << "-------------> Add New Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID: ";

cin >> slotId;

cout << "Enter Slot Location: ";

cin.ignore();

getline(cin, location);

ofstream out("parkingslots.txt", ios::app);

out << "Slot ID: " << slotId << ", Location: " << location << ", Status: Available\n";

out.close();

cout << "Parking slot added successfully!\n";

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::viewAllParkingSlots()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> View All Parking Slots <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("parkingslots.txt");

string line;

while (getline(in, line))

{

cout << line << endl;

}

in.close();

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::removeParkingSlot()

{

system("CLS");

string slotId, line;

cout << "---------------------------------------------------\n";

cout << "-------------> Remove Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to remove: ";

cin >> slotId;

ifstream in("parkingslots.txt");

ofstream out("temp.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

found = true;

cout << "Removing: " << line << endl;

}

else

{

out << line << endl;

}

}

in.close();

out.close();

remove("parkingslots.txt");

rename("temp.txt", "parkingslots.txt");

if (found)

{

cout << "Parking slot removed successfully!\n";

}

else

{

cout << "Slot ID not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::viewVehicleHistory()

{

system("CLS");

string line, vehicleid;

cout << "---------------------------------------------------\n";

cout << "-------------> View Vehicle History <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to view history: ";

cin >> vehicleid;

ifstream in("vehicle\_history.txt");

if (!in)

{

cout << "Failed to open vehicle\_history.txt.\n";

return;

}

bool found = false;

while (getline(in, line))

{

if (line.find(vehicleid) != string::npos)

{

cout << line << endl;

found = true;

}

}

in.close();

if (!found)

{

cout << "No history found for Vehicle ID " << vehicleid << ".\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::viewStatistics()

{

cout << "---------------------------------------------------\n";

cout << "-------------> View System Statistics <-------------\n";

cout << "---------------------------------------------------\n";

cout << "1. Total Vehicles Parked Today\n";

cout << "2. Revenue Generated\n";

cout << "3. Average Parking Time" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "Total Vehicles Parked Today: 50\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Revenue Generated Today: $500\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "Average Parking Time: 2 hours\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::help()

{

cout << "---------------------------------------------------\n";

cout << "---------------- Help & FAQs ------------------\n";

cout << "---------------------------------------------------\n";

cout << "1. How to park a vehicle\n";

cout << "2. How to remove a vehicle\n";

cout << "3. Contact support" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "To park a vehicle, follow the steps:\n";

cout << "1. Choose a slot.\n";

cout << "2. Enter vehicle details.\n";

cout << "3. Confirm reservation.\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "To remove a vehicle, please contact the management team with your vehicle details.\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "Contact support at support@parking.com or call 123-456-7890.\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::emergencyAlert()

{

cout << "---------------------------------------------------\n";

cout << "-------------> Emergency Alert <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Emergency reported! Notifying security and management...\n";

cout << "Emergency alert sent. Please stay calm.\n";

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::notifyOverduePayment()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> Notify Overdue Payment <-------------\n";

cout << "---------------------------------------------------\n";

int overdueThreshold = 24;

string vehicleId;

cout << "Enter Vehicle ID to check overdue payment: ";

cin >> vehicleId;

ifstream in("vehicle\_payment\_info.txt");

bool found = false;

string line;

while (getline(in, line))

{

if (line.find(vehicleId) != string::npos)

{

cout << "\nReminder: Vehicle " << vehicleId << " has overdue payment!\n";

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "\nPayment is up to date.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

cin.ignore();

cin.get();

}

void ParkingManagement2::settings()

{

cout << "---------------------------------------------------\n";

cout << "-------------------- System Settings ----------------\n";

cout << "---------------------------------------------------\n";

cout << "1. Change Language\n";

cout << "2. Update Notification Preferences\n";

cout << "3. Reset System" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "Language settings: English (default).\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Notification preferences updated: Email and SMS notifications enabled.\n";

cout << "---------------------------------------------------\n";

break;

case 3:

cout << "---------------------------------------------------\n";

cout << "System reset completed. All data erased.\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::adminMode()

{

cout << "---------------------------------------------------\n";

cout << "-------------> Admin Mode <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Admin Options:\n";

cout << "1. Reset System Logs\n";

cout << "2. Manage Parking Slots" << endl;

int choice;

cin >> choice;

cin.ignore();

switch (choice)

{

case 1:

cout << "---------------------------------------------------\n";

cout << "System logs reset. All data cleared.\n";

cout << "---------------------------------------------------\n";

break;

case 2:

cout << "---------------------------------------------------\n";

cout << "Managing parking slots...\n";

cout << "---------------------------------------------------\n";

break;

default:

cout << "Invalid option. Try again.\n";

break;

}

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::park()

{

cout << "---------------------------------------------------\n";

cout << "-------------> Vehicle Parking System <-------------\n";

cout << "---------------------------------------------------\n";

ofstream out;

vehicle VEHICLE[10];

out.open("vehicleparkdb.txt", ios::app);

cout << "Enter Your Name: ";

cin.ignore();

getline(cin, VEHICLE[count].drivername);

cout << "Enter Your Vehicle Name: ";

getline(cin, VEHICLE[count].vehiclename);

cout << "Enter Vehicle Registration Number: ";

cin >> VEHICLE[count].vehicleid;

cout << "Enter Your Time Stay In Hours (Integer Only): ";

cin >> VEHICLE[count].timestay;

out << "Driver Name: " << VEHICLE[count].drivername << "\n"

<< "Vehicle Name: " << VEHICLE[count].vehiclename << "\n"

<< "Vehicle ID: " << VEHICLE[count].vehicleid << "\n"

<< "Time Stay: " << VEHICLE[count].timestay << " hours\n";

cout << "Your Vehicle Has Been Parked!" << endl;

out.close();

count++;

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::viewAllVehiclesNeedingCharging()

{

system("CLS");

cout << "---------------------------------------------------\n";

cout << "-------------> View All Vehicles Needing Charging <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("vehicleparkdb.txt");

if (!in)

{

cout << "Error opening file!" << endl;

return;

}

string line;

bool found = false;

while (getline(in, line))

{

if (line.find("Requires Charging: Yes") != string::npos) {

cout << "\nVehicle requiring charging: \n";

cout << line << endl;

found = true;

}

}

in.close();

if (!found)

{

cout << "\nNo vehicles require charging at the moment.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

cin.ignore();

cin.get();

}

void ParkingManagement2::searchByDriverName()

{

system("CLS");

string driverName;

string line;

bool found = false;

cout << "Enter the Driver's Name: ";

getchar();

getline(cin, driverName);

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

ifstream in;

in.open("vehicleparkdb.txt");

while (getline(in, line))

{

if (line.find(driverName) != string::npos)

{

cout << "Vehicle Found!" << endl;

cout << line << endl;

getline(in, line);

cout << line << endl;

found = true;

break;

}

}

in.close();

if (!found)

{

cout << "No vehicle found for the driver name: " << driverName << endl;

}

cout << "Press Enter Key To Return" << endl;

getchar();

}

void ParkingManagement2::displayParkingRules()

{

system("CLS");

cout << " -------------> Parking Rules <-------------" << endl

<< endl;

cout << "1. Each vehicle must be assigned a designated parking slot." << endl;

cout << "2. Vehicles must adhere to the time limit specified." << endl;

cout << "3. Different parking fees apply based on vehicle type." << endl;

cout << "4. Payment must be made in advance or upon exiting." << endl;

cout << "5. Check availability before arriving at the facility." << endl;

cout << "6. Only roadworthy vehicles are allowed." << endl;

cout << "7. Electric vehicles must use designated charging stations." << endl;

cout << "8. Follow emergency protocols in case of emergencies." << endl;

cout << "9. Feedback and complaints should be directed to management." << endl;

cout << "10. The facility is monitored for security." << endl;

cout << "11. Prohibited activities include smoking and unauthorized repairs." << endl;

cout << "12. Maintenance may affect parking availability." << endl;

cout << "13. Compliance with rules is mandatory." << endl;

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

cout << "Press Enter Key To Return" << endl;

cin.get();

cin.ignore();

}

void ParkingManagement2::sendReminderForPayment()

{

string vehicleid;

cout << "---------------------------------------------------\n";

cout << "-------------> Send Reminder For Payment <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Reminder sent to the owner of vehicle ID: " << vehicleid << endl;

cin>>vehicleid;

cout<<"Reminder is sent to the owner of vehicle ID= "<<vehicleid<<endl;

cout << "---------------------------------------------------\n";

cin.ignore();

cin.get();

}

void ParkingManagement2::generateExpenseReport()

{

system("CLS");

string line;

cout << "---------------------------------------------------\n";

cout << "-------------> Generate Expense Report <-------------\n";

cout << "---------------------------------------------------\n";

ifstream in("expenses.txt");

double totalExpense = 0.0;

while (getline(in, line))

{

stringstream ss(line);

string word;

double expenseAmount = 0.0;

while (ss >> word)

{

expenseAmount = stod(word);

}

totalExpense += expenseAmount;

}

in.close();

cout << "Total Expenses: $" << totalExpense << endl;

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

bool ParkingManagement2::viewParkingSlotAvailability()

{

ifstream in("parkingslots.txt");

string line;

int slotId;

while (getline(in, line))

{

if (line.find(to\_string(slotId)) != string::npos)

{

if (line.find("Available") != string::npos)

{

cout << "Slot ID " << slotId << " is available.\n";

return true;

}

else

{

cout << "Slot ID " << slotId << " is not available.\n";

return false;

}

}

}

in.close();

cout << "Slot ID " << slotId << " not found.\n";

return false;

}

void ParkingManagement2::findParkingSlotByID()

{

int slotID;

if (slotID >= 1 && slotID <= parkingslots.size())

{

cout << "Parking Slot for ID " << slotID << " is: " << parkingslots[slotID - 1] << endl;

}

else

{

cout << "Invalid Parking ID! Please enter a valid ID between 1 and " << parkingslots.size() << "." << endl;

}

}

void ParkingManagement2::allocateParkingSlot()

{

system("CLS");

string slotId, vehicleId, line, updatedLine;

cout << "---------------------------------------------------\n";

cout << "-------------> Allocate Parking Slot <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Slot ID to allocate: ";

cin >> slotId;

cout << "Enter Vehicle ID: ";

cin >> vehicleId;

ifstream in("parkingslots.txt");

ofstream out("temp.txt");

bool found = false;

while (getline(in, line))

{

if (line.find(slotId) != string::npos)

{

found = true;

updatedLine = line + ", Vehicle ID: " + vehicleId;

out << updatedLine << endl;

}

else

{

out << line << endl;

}

}

in.close();

out.close();

remove("parkingslots.txt");

rename("temp.txt", "parkingslots.txt");

if (found)

{

cout << "Parking slot allocated successfully!\n";

}

else

{

cout << "Slot ID not found.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

void ParkingManagement2::expense()

{

system("CLS");

string cexpenseid, line;

int i = 0;

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

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

cout << endl;

cout << " -------------> Vehicle Parking Expenses <-------------" << endl;

cout << endl;

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

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

cout << "Enter Your Vehicle Id : ";

cin >> cexpenseid;

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

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

ifstream in;

in.open("vehicleparkexpensedb.txt");

while (in.eof() == 0)

{

getline(in, line);

size\_t found = line.find(cexpenseid);

if (found != string::npos)

{

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

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

cout << "Your Vehicle Parking Expense Details Are : " << endl;

cout << line;

getline(in, line);

cout << endl;

cout << line << endl;

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

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

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

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

cout << "Select payment method: " << endl;

cout << "1. Credit Card" << endl;

cout << "2. Debit Card" << endl;

cout << "3. Cash" << endl;

cout << "4. Digital Wallet" << endl;

int paymentChoice;

cin >> paymentChoice;

switch (paymentChoice)

{

case 1:

cout << "Payment via Credit Card selected." << endl;

break;

case 2:

cout << "Payment via Debit Card selected." << endl;

break;

case 3:

cout << "Payment via Cash selected." << endl;

break;

case 4:

cout << "Payment via Digital Wallet selected." << endl;

break;

default:

cout << "Invalid choice! Please select a valid payment method." << endl;

break;

}

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

in.close();

i++;

break;

}

}

in.close();

if (i == 0)

{

cout << "Vehicle With VehicleID " << cexpenseid << " Not Found" << endl;

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

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

cout << "TRY AGAIN! (Use Correct VehicleID)" << endl;

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

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

cout << "Press Enter Key To Return" << endl;

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

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

getch();

}

}

void ParkingManagement::addParkingPlace()

{

string newparkingPlace;

cout << "Enter the name of the parking slot to add: ";

getline(cin,newparkingPlace);

cin.ignore(10000, '\n');

parkingslots.push\_back(newparkingPlace);

cout << "Parking slot '" << newparkingPlace << "' has been added.\n";

}

void ParkingManagement::modifyReservation() {

system("CLS");

string vehicleId;

cout << "---------------------------------------------------\n";

cout << "-------------> Modify Reservation <-------------\n";

cout << "---------------------------------------------------\n";

cout << "Enter Vehicle ID to modify reservation: ";

cin >> vehicleId;

ifstream in("reservations.txt");

ofstream out("temp.txt");

string line;

bool found = false;

while (getline(in, line)) {

if (line.find(vehicleId) != string::npos) {

found = true;

cout << "Current Reservation: " << line << endl;

// Get new parking slot

cout << "Enter new Parking Slot: ";

string newParkingSlot;

cin.ignore();

getline(cin, newParkingSlot);

// Modify the reservation line

line.replace(line.find("Parking Slot: ") + 14, line.find(", Reservation Time:") - (line.find("Parking Slot: ") + 14), newParkingSlot);

out << line << endl; // Write modified line to temp file

} else {

out << line << endl; // Write other reservations to temp file

}

}

in.close();

out.close();

remove("reservations.txt");

rename("temp.txt", "reservations.txt");

if (found) {

cout << "Reservation modified successfully!\n";

} else {

cout << "Vehicle ID not found in reservations.\n";

}

cout << "---------------------------------------------------\n";

cout << "Press Enter Key To Return\n";

getchar();

}

int main()

{

system("CLS");

login loginobj;

loginobj.loginsys();

ParkingManagement parkobj;

vector<string> parkingslots = {

"Downtown Parking",

"Central Park Garage",

"City Hall Lot",

"Mall Parking",

"Airport Parking",

"Train Station Parking",

"Beach Parking",

"Stadium Parking",

"Library Parking",

"Museum Parking",

"Hospital Parking",

"University Parking",

"Shopping Center Parking",

"Residential Area Parking",

"Business District Parking",

"Cinema Parking",

"Zoo Parking",

"Park and Ride",

"Sports Complex Parking",

"Convention Center Parking",

"Hotel Parking",

"Theater Parking",

"Restaurant Parking",

"Night Club Parking",

"Market Parking",

"Church Parking",

"Temple Parking",

"Mosque Parking",

"School Parking",

"College Parking",

"Playground Parking",

"Community Center Parking",

"Fire Station Parking",

"Police Station Parking",

"Government Office Parking",

"Bus Terminal Parking",

"Taxi Stand Parking",

"Ferry Terminal Parking",

"Bridge Parking",

"Tunnel Parking",

"Court House Parking",

"Post Office Parking",

"Swimming Pool Parking",

"Gym Parking",

"Recreation Center Parking",

"Amusement Park Parking",

"Water Park Parking",

"Casino Parking",

"Race Track Parking",

"Botanical Garden Parking"

};

while (1)

{

system("CLS");

int choice;

cout << " -------------> Vehicle Parking System <-------------" << endl

<< endl;

#ifdef \_WIN32

Sleep(2000); // Sleep for 2 seconds on Windows

#else

sleep(2); // Sleep for 2 seconds on Unix-like systems

#endif

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

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

cout << "1. Park Your Vehicle" << endl;

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

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

cout << "2. View Your Vehicle Details" << endl;

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

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

cout << "3. Check Expenses" << endl;

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

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

cout << "4. Remove Your Vehicle" << endl;

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

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

cout << "5. Provide Feedback" << endl;

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

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

cout << "6. View All Parked Vehicles" << endl;

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

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

cout << "7. Edit Vehicle Details" << endl;

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

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

cout << "8. Search Parking Place" << endl;

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

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

cout << "9. View All Feedback" << endl;

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

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

cout << "10. Generate Report" << endl;

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

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

cout << "11. Add New Parking Slot" << endl;

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

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

cout << "12. View All Parking Slots" << endl;

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

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

cout << "13. Remove Parking Slot" << endl;

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

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

cout << "14.View Charging vehicles" << endl;

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

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

cout << "15.Online authorisation" << endl;

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

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

cout << "16. Edit Parking Slot Details" << endl;

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

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

cout << "17.view Vehicle Details By vehicle Id" << endl;

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

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

cout << "18. list of Vehicles Type" << endl;

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

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

cout << "19.add Vehicle History " << endl;

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

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

cout << "20.view Vehicle History " << endl;

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

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

cout << "21.Send Reminder For Payment" << endl;

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

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

cout << "22.Find parking slot by id" << endl;

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

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

cout << "23.view statistics " << endl;

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

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

cout << "24.help " << endl;

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

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

cout << "25.settings " << endl;

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

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

cout << "26.admin mode" << endl;

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

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

cout << "27.Notification for exceeding time" << endl;

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

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

cout << "28.search By Driver Name " << endl;

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

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

cout << "29.emergency alert " << endl;

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

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

cout << "30.Add parking place" << endl;

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

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

cout << "31.Display Parking rules " << endl;

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

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

cout << "32.Reservation " << endl;

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

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

cout << "33.View all reserved vehicles " << endl;

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

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

cout << "34.Cancel reservations " << endl;

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

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

cout << "35.Check status of reservation " << endl;

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

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

cout << "36.Modify reservation " << endl;

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

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

cout << "37.Exit " << endl;

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

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

cout << "ENTER YOUR CHOICE : ";

cin >> choice;

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

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

switch (choice)

{

case 1:

char yn;

cout << "Do You Want To Continue (y/n) : ";

cin >> yn;

system("CLS");

if (yn == 'y')

{

parkobj.park(parkobj);

}

system("CLS");

break;

case 2:

parkobj.vehicledetail();

break;

case 3:

parkobj.expense();

break;

case 4:

parkobj.removevehicle();

break;

case 5:

parkobj.getfeedback();

break;

case 6:

parkobj.viewAllParkedVehicles();

break;

case 7:

parkobj.editVehicleDetails();

break;

case 8:

parkobj.searchParkingPlace();

break;

case 9:

parkobj.viewAllFeedback();

break;

case 10:

parkobj.generateReport();

break;

case 11:

parkobj.addParkingSlot();

break;

case 12:

parkobj.viewAllParkingSlots();

break;

case 13:

parkobj.removeParkingSlot();

break;

case 14:

parkobj.viewAllVehiclesNeedingCharging();

break;

case 15:

onlineAuthorization();

break;

case 16:

parkobj.editParkingSlotDetails();

break;

case 17:

parkobj.viewVehicleDetailsById();

break;

case 18:

parkobj.listVehiclesByType();

break;

case 19:

parkobj.addVehicleHistory();

break;

case 20:

parkobj.viewVehicleHistory();

break;

case 21:

parkobj.sendReminderForPayment();

break;

case 22:

parkobj.findParkingSlotByID();

break;

case 23:

parkobj.viewStatistics();

break;

case 24:

parkobj.help();

break;

case 25:

parkobj.settings();

break;

case 26:

parkobj.adminMode();

break;

case 27:

parkobj.notifyOverduePayment();

break;

case 28:

parkobj.searchByDriverName();

break;

case 29:

parkobj.emergencyAlert();

break;

case 30:

parkobj.addParkingPlace();

break;

case 31:

parkobj.displayParkingRules();

break;

case 32:

parkobj.reserveParkingSpot();

break;

case 33:

parkobj.viewAllReservedVehicles();

break;

case 34:

parkobj.CancelReservedVehicle();

break;

case 35:

parkobj.checkReservationStatus();

break;

case 36:

parkobj.modifyReservation();

break;

case 37:

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

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

cout << "Press Enter Key To EXIT";

getch();

exit(0);

default:

system("CLS");

cout << "You Entered A Wrong Choice!" << endl;

break;

}

}

return 0;

}