Jaypee Institute of Information Technology
Sector-62, Noida

**A**
**Project Report**
**Of**
**Software development Fundamentals Lab - II**

# SDF MINI PROJECT REPORT

# Title: WHEELS BUDDY

## "An Integrated Carpooling and Expense Tracking System"

**Batch: B10 (B.Tech CSE)**
**TEAM MEMBERS:**

| S. No. | Enrolment No. | Name |
|---|---|---|
| 1 | 23103298 | Aaditya Pratap Singh |
| 2 | 23103278 | Kush Kansal |
| 3 | 23103292 | Sriyash Mishra |

# SYNOPSIS-:

**Problem Statement:** **In today's fast-paced world, daily commutes often result in inefficiency, environmental strain, and isolated travel experiences. To tackle these challenges effectively, there's a crucial demand for a one stop solution that seamlessly blends carpooling with meticulous expense management. Our project, Wheels Buddy, is poised to revolutionize commuting by presenting a user-centric platform. Our aim is to foster community building, convenience, and eco-consciousness.**

**ABSTRACT-** **Introducing Wheels Buddy – your all-in-one solution for seamless carpooling and expense tracking. With a focus on convenience, efficiency, and community building, Wheels Buddy revolutionizes the way you commute by merging carpooling with meticulous expense management. Our platform offers a comprehensive suite of features, ranging from user profiles and dynamic route planning to expense sharing and verification systems. By facilitating resource-sharing and reducing environmental impact, Wheels Buddy simplifies your daily travels and fosters a sense of camaraderie among users**

## PROJECT OBJECTIVES:

**1. Develop a user-friendly interface for Wheels Buddy, enabling seamless profile creation, route planning, ride offering/finding, and expense tracking.**
**2. Implement a robust expense tracking system to record shared expenses accurately.**
**3. Establish secure verification processes for user trust.**
**4. Optimize features based on user feedback for enhanced experience.**

## OUR PROJECT BOASTS SOME KEY FEATURES-:

1. User Profiles: - Users can create profiles with information such as name, contact details, and preferences.

2. Car Information: - Display a list of available cars with details on the model, seating capacity, and driver information.

3. Route Planning: - Allow users to plan routes for their journeys, specifying pick-up and drop-off points.

4. Finding a Ride: - Enable users to search for available rides based on their desired routes and timings.

5. Ride Booking: - Implement a system for users to book available seats in a carpool.

6. Expense Sharing: - Introduce a mechanism for sharing expenses among participants, considering factors like distance, fuel costs, and tolls.

7. Expense Tracking: - Record and categorize shared expenses, including fuel and toll charges. Also, generate a bill.

8. Review and Rating System: - Implement a review and rating system for both drivers and passengers to build a trustworthy carpooling community.

9. User Dashboard: - Provide users with a dashboard displaying their upcoming rides, ride history, and overall expenses.

10. Security and Verification: - Implement a verification system for users to enhance trust within the carpooling community.

# FEATURES OF C++ USED-:

- **File Handling:** Reading and writing data to files to store information about available cars, bookings, and expenses.
- **Object-Oriented Programming (OOP):** Designing classes and objects to represent entities such as cars, drivers, and users.
- Implementing encapsulation, inheritance, and polymorphism for code organization and reusability.
- **Functions:** Defining functions for specific tasks like booking a car, calculating expenses, etc.
- **Conditional Statements and Loops:** Using if statements and loops for decision-making and iterative processes in various parts of the program.
- **Dynamic Memory Allocation:** Allocating and deallocating memory dynamically as needed.
- **Exception Handling:** Implementing error handling mechanisms to manage unexpected situations or invalid inputs.
- **String Manipulation:** Performing operations on strings, such as concatenation or substring extraction, for handling textual data.
- **User Interface (Console-based):** Creating a user-friendly interface using cout and cin for displaying information, taking user input, and providing a smooth experience.
- **STL (Standard Template Library):** Using additional STL features and algorithms for enhanced efficiency and code readability in various parts of the project.

# IMPLEMENTATION CODE

```
/**
 * Includes necessary libraries and declares the Pages class.
 */
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <windows.h> //for usleep function
#include <stdio.h>   // for mkdir
#include <io.h>      //for mkdir
#include <iomanip>   //for setw
#include <sstream>   // FOR TOSTRING FUNCTION() OF RIDE CLASS
#include <cstdlib>   //RANDOM NUMBER
#include <ctime>     //RANDOM NUMBER
#include <algorithm>
#include <conio.h>
using namespace std;
class Pages; // Forward declaration
class User
{
private:
    string username;
    string password;
public:
    User(string uname, string pwd)
    {
        username = uname;
        password = pwd;
    }
    string getUsername()
    {
        return username;
    }
    string getPassword()
    {
        return password;
    }
    bool checkCredentials(string uname, string pwd)
    {
        return (username == uname && password == pwd);
    }
};
class LoginManager
{
private:
    vector<User> users;
public:
    /**
     * Constructs a LoginManager object by reading user data from a file.
     * @param filename The name of the file containing user data.*/
    LoginManager(string filename)
    {
```

```
    ifstream fileIn(filename);
    if (!fileIn.is_open())
    {
        cerr << "Error: Unable to open file for user data." << endl;
        return;
    }
    string username, password;
    while (fileIn >> username >> password)
        users.push_back(User(username, password));
    fileIn.close();
}
bool authenticate(string uname, string pwd)
{
    for (auto &user : users)
    {
        if (user.checkCredentials(uname, pwd))
        {
            return true;
        }
    }
    return false;
}
/**
 * Displays a login screen for user authentication.
 * @return A pointer to a User object representing the logged-in user.*/
User *loginScreen()
{
    system("cls");
    system("Color A0");
    cout << "****************************************\n";
    cout << "*                Login               *\n";
    cout << "****************************************\n";
    string username, password;
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    char s[20] = {0};
    int i;
    for (i = 0; i < 20; i++)
    {
        s[i] = _getch();
        if (s[i] == 13)
            break;
        _putch('*');
    }
    s[i] = '\0'; // Null-terminate the character array
    password = s;
    cout << endl;
    User *ob = new User(username, password);
    if (authenticate(username, password))
        return ob;
    else
    {
        User *o = new User("null", "null");
        return o;
```

```
        }
    }
};
class RegistrationManager
{
public:
    /**
     * Registers a new user by creating an account with a username and password.
     * @param filename The name of the file to store user information.*/
    static void registerUser(string &filename)
    {
        system("cls");
        system("color A1");
        cout << "****************************************\n";
        cout << "*         CREATE A NEW ACCOUNT          *\n";
        cout << "****************************************\n";
        string username, password;
        cout << "Enter a new username: ";
        cin >> username;–
        cout << "Enter a new password: ";
        cin >> password;
        ofstream fileOut(filename, ios::app);
        if (fileOut.is_open())
        {
            fileOut << username << " " << password << endl;
            fileOut.close();
            cout << "Account created successfully!\n";
            createFolder(username);
            // A important step
        }
        system("cls");
    }
    /**
     * Creates a folder with the given name and initializes necessary files within it.
     * @param folderName The name of the folder to be created.*/
    static void createFolder(string folderName)
    {
        // Create the folder
        mkdir(("./Files/" + folderName).c_str());
        // Create files inside the folder
        string folderPath = "./Files/" + folderName;
        ofstream pastRidesFile(folderPath + "/pastRides.txt");
        ofstream upcomingRidesFile(folderPath + "/upcomingRides.txt");
        ofstream userDetailsFile(folderPath + "/userDetails.txt");
       pastRidesFile << "| Ride ID    | Date       | Time   | Source City  | Destination City  | Max Passengers | Current
Passengers  | Car Model     | Fare(Rs) | Distance (km) |" << endl;
        pastRidesFile << "|------------|------------|--------|--------------|-------------------|----------------|--------------------|------
---------|----------|---------------|" << endl;
        upcomingRidesFile << "| Ride ID    | Date       | Time   | Source City  | Destination City  | Max Passengers |
Current Passengers  | Car Model     | Fare(Rs) | Distance (km) |" << endl;
        upcomingRidesFile << "|------------|------------|--------|--------------|-------------------|----------------|--------------------
|----------------|----------|---------------|" << endl;
        // Close the files
        pastRidesFile.close();
        upcomingRidesFile.close();
```

```
        string username, fullName, age, gender, email, address, phone, aadharNo, memberSince, ridesTaken,
amountSpent;
        username = folderName;
        cout << "Enter Full Name: ";
        getline(cin, fullName);
        getline(cin, fullName);
        cout << "Enter Age: ";
        getline(cin, age);
        cout << "Enter Gender: ";
        getline(cin, gender);
        cout << "Enter Email: ";
        getline(cin, email);
        cout << "Enter Address: ";
        getline(cin, address);
        cout << "Enter Phone: ";
        getline(cin, phone);
        cout << "Enter Aadhar Card No: ";
        getline(cin, aadharNo);
        // Write user details to file
        userDetailsFile << "Username: " << username << endl;
        userDetailsFile << "Full Name: " << fullName << endl;
        userDetailsFile << "Age: " << age << endl;
        userDetailsFile << "Gender: " << gender << endl;
        userDetailsFile << "Email: " << email << endl;
        userDetailsFile << "Address: " << address << endl;
        userDetailsFile << "Phone: " << phone << endl;
        userDetailsFile << "Aadhar Card No: " << aadharNo << endl;
        userDetailsFile << "Member Since: " << memberSince << endl;
        userDetailsFile << "Total Rides Taken: " << 0 << endl;
        // Close the file
        userDetailsFile.close();
    }
};
class Pages // Define Pages class before main
{
public:
    void fileLoadingPage()
    {
        system("cls");
        system("color 6B");
        ifstream in("CodeRelatedFiles/Welcome.txt"); // displaying welcome ASCII image text on output
screen fn1353
        char str[1000];
        while (in)
        {
            in.getline(str, 1000); // delim defaults to '\n' cp
            if (in)
                cout << str << endl;
        }
       in.close();
        Sleep(500);
        cout << "\nStarting the program please wait....." << endl;
        Sleep(500);
        cout << "\nloading up files....." << endl;
        Sleep(500);
```

```
    system("cls");
}
// this function returns the choice of the user
/**
 * Displays the home page of the Wheel Buddy application and prompts the user for a choice.
 * @return The user's choice selected from the home page menu.*/
int homePage()
{
    system("color 0A");
    cout << "\n _____________________________________________________________________________\n";
    cout << "||                                                           ||\n";
    cout << "||   \\\      /\\\      / |====== |      |====== /=====\\\ |\\\    /| |=====    ||\n";
    cout << "||    \\\    /  \\\    /  |_____ |      |      |     | | \\\  / | |_____    ||\n";
    cout << "||     \\\  /    \\\  /   |      |      |      |     | |  \\\/  | |         ||\n";
    cout << "||      \\\/      \\\/    |====== |_____ |====== \\\=====/ |     | |=====    ||\n";
    cout << "||                          TO WHEEL BUDDY                               ||\n";
    cout << "||    MENU:-                                                 ||\n";
    cout << "||   1. Login                                                ||\n";
    cout << "||   2. Create a new account                                    ||\n";
    cout << "||   3. CLOSE WHEELBUDDY                                        ||\n";
    cout << "||_____________________________________________________________________________||\n";
    cout << "ENTER YOUR CHOICE: ";
    int choice;
    cin >> choice;
    cout << "\nLoading.....\n";
    Sleep(1000);
    return choice;
}
/**
 * Displays a thank you message and prompts the user to confirm exit.
 * If confirmed, displays a goodbye message, plays a video, and exits the program.
 * If not confirmed, exits the program.*/
void thankYouPage()
{
    cout << "\nAre you sure you want to exit? (YES/NO)---";
    string ch;
    cin >> ch;
    cin.ignore();
    if (ch == "YES" || ch == "yes" || ch == "Yes")
    {
        system("cls");
       system("color 6D");
       cout << " ________  __                  __          __      __                __ __ \n";
       cout << "/        |/  |                /  |        /  \\    /  |              /  |/  |\n";
       cout << "$$$$$$$$/ $$ |____   ______   _______   $$ |  __       $$  \\  /$$/_____  __  __       $$ |$$ |\n";
       cout << "   $$ |   $$      \\ /      \\ /       \\ $$ | /  |      $$  \\/$$//      \\ / |  / |      $$ |$$ |\n";
       cout << "   $$ |   $$$$$$$  |$$$$$$  |$$$$$$$  |$$ |_/$$/        $$  $$//$$$$$$  |$$ |  $$ |      $$ |$$ |\n";
       cout << "   $$ |   $$ |  $$ | /    $$ |$$ |  $$ |$$   $$<          $$$$/ $$ |  $$ |$$ |  $$ |      $$/ $$/ \n";
       cout << "   $$ |   $$ |  $$ |/$$$$$$$ |$$ |  $$ |$$$$$$  \\          $$ | $$ \\__$$ |$$ \\__$$ |       __  __  \n";
       cout << "   $$ |   $$ |  $$ |$$    $$ |$$ |  $$ |$$ | $$  |         $$ | $$    $$/ $$    $$/       /  |/  |\n";
       cout << "   $$/    $$/   $$/  $$$$$$$/ $$/   $$/ $$/   $$/          $$/   $$$$$$/   $$$$$$/        $$/ $$/ \n";
       Sleep(2000);
        cout << "\n\n";
        system("color 5E");
        Sleep(400);
        system("color 46");
        Sleep(400);
```

```
            cout << "\nGoodbye! Have a great day!\n";
            system("cls");
            system("start vlc --fullscreen TMKOC.mp4");
            exit(0);
        }
        else
        {
            system("cls");
            exit(0);
        }
    }
};

class Dashboard : public Pages
{
public:
    /**
     * Displays the user dashboard with user details and upcoming rides.
     * @param username The username of the user to display the dashboard for.*/
    void display(string username)
    {
        system("color E4");
        cout << "*************************** User Dashboard ***************************" <<
endl;
        cout << "User Details:" << endl;
        string filename = "Files/" + username + "/userDetails.txt";
        ifstream file(filename);
        string line;
        while (getline(file, line))
        {
            cout << "\t  " << line << endl;
        }
        cout << "-------------------------------------------------------------------\n"
             << endl;
        file.close();
        cout << "Past Rides Details:" << endl;
        cout << "\t-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
        filename = "Files/" + username + "/pastRides.txt";
        ifstream file2(filename);
        while (getline(file2, line))
        {
            cout << "\t" << line << endl;
        }
        cout << "\t-------------------------------------------------------------------------------------------------------------------------------------------\n"
             << endl;
        file2.close();
        cout << "Upcoming Rides Details:" << endl;
        cout << "\t-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
        filename = "Files/" + username + "/upcomingRides.txt";
        ifstream file3(filename);
        while (getline(file3, line))
        {
```

```
            cout << "\t" << line << endl;
        }
        cout << "\t-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------\n"
             << endl;
        file3.close();
    }
};
class Ride
{
private:
    string rideID;
    string date;
    string time;
    string sourceCity;
    string destinationCity;
    int maxPassengers;
    int currentPassengers;
    string carModel;
    double fare;
    double distance;
public:
    /**
     * Constructs a Ride object from a string input.
     * @param s The string containing ride information separated by '|'.*/
    Ride(string s)
    {
        stringstream ss(s);
        string temp;
        getline(ss, temp, '|'); // Skip first |
        ss >> ws;               // Skip whitespaces
        getline(ss, rideID, '|');
        ss >> ws; // Skip whitespaces
        getline(ss, date, '|');
        ss >> ws; // Skip whitespaces
        getline(ss, time, '|');
        ss >> ws; // Skip whitespaces
        getline(ss, sourceCity, '|');
        ss >> ws; // Skip whitespaces
        getline(ss, destinationCity, '|');
        ss >> ws; // Skip whitespaces
        getline(ss, temp, '|');
        temp = trim(temp);
        try
        {
            maxPassengers = stoi(temp);
        }
        catch (const invalid_argument &e)
        {
        }
        ss >> ws; // Skip whitespaces
        getline(ss, temp, '|');
        try
        {
            currentPassengers = stoi(temp);
```

```
    }
    catch (const invalid_argument &e)
    {
    }
    ss >> ws; // Skip whitespaces
    getline(ss, carModel, '|');
    ss >> ws; // Skip whitespaces
    getline(ss, temp, '|');
    temp = trim(temp);
    try
    {
        fare = stod(temp);
    }
    catch (const invalid_argument &e)
    {
    }
    ss >> ws; // Skip whitespaces
    getline(ss, temp, '|');
    temp = trim(temp);
    try
    {
        distance = stod(temp);
    }
    catch (const invalid_argument &e)
    {
    }
    // Convert string values to integer or double
}

/**
 * Constructs a Ride object with the provided details.
 * @param date The date of the ride.
 * @param time The time of the ride.
 * @param sourceCity The source city of the ride.
 * @param destinationCity The destination city of the ride.
 * @param maxPassengers The maximum number of passengers allowed for the ride.
 * @param currentPassengers The current number of passengers in the ride.
 * @param carModel The model of the car for the ride.
 * @param fare The fare for the ride.
 * @param distance The distance of the ride.
 * @returns A Ride object with the specified details.*/
Ride(string date, string time, string sourceCity, string destinationCity,
     int maxPassengers, int currentPassengers, string carModel, double fare, double distance)
{
    this->rideID = generateRandomRideID();
    this->date = date;
    this->time = time;
    this->sourceCity = sourceCity;
    this->destinationCity = destinationCity;
    this->maxPassengers = maxPassengers;
    this->currentPassengers = currentPassengers;
    this->carModel = carModel;
    this->fare = fare;
    this->distance = distance;
}
```

```
    string getRideID() { return trim(rideID); }
    string getDate() { return trim(date); }
    string getTime() { return trim(time); }
    string getSourceCity() { return trim(sourceCity); }
    string getDestinationCity() { return trim(destinationCity); }
    int getMaxPassengers() { return maxPassengers; }
    int getCurrentPassengers() { return currentPassengers; }
    string getCarModel() { return trim(carModel); }
    double getFare() { return fare; }
    double getDistance() { return distance; }
    void setCurrentPassengers(int passengers)
    {
        currentPassengers = passengers;
    }
    /**
     * Trims leading and trailing whitespace characters from a given string.
     * @param str The input string to be trimmed.
     * @returns The trimmed string with leading and trailing whitespace removed.*/
    string trim(const string &str)
    {
        size_t firstNonSpace = str.find_first_not_of(" \t\n\r"); // Find index of first non-whitespace character
        size_t lastNonSpace = str.find_last_not_of(" \t\n\r");   // Find index of last non-whitespace character
        if (firstNonSpace == string::npos || lastNonSpace == string::npos)
        {
            // Handle the case when the string is empty or contains only whitespace
            return "";
        }
        else
        {
            // Return the substring between the first and last non-whitespace characters
            return str.substr(firstNonSpace, lastNonSpace - firstNonSpace + 1);
        }
    }
    string generateRandomRideID()
    {
        string alphanumeric =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
        int idLength = 5;
        string id;
        srand(std::time(0)); // Specify the namespace for time()

        for (int i = 0; i < idLength; ++i)
        {
            id += alphanumeric[rand() % alphanumeric.length()];
        }
        return id;
    }
    string toString()
    {
        stringstream ss;
        ss << "| " << setw(11) << left << truncate(rideID, 11) << "| "
           << setw(12) << left << truncate(date, 12) << "| "
           << setw(7) << left << truncate(time, 7) << "| "
           << setw(13) << left << truncate(sourceCity, 13) << "| "
           << setw(18) << left << truncate(destinationCity, 18) << "| "
```

```
            << setw(15) << left << maxPassengers << "| "
            << setw(20) << left << currentPassengers << "| "
            << setw(14) << left << truncate(carModel, 14) << "| "
            << setw(9) << left << fare << "| "
            << setw(14) << left << distance << "|";
        return ss.str();
    }
    // Helper function to truncate strings longer than a specified length
    string truncate(string str, size_t width)
    {
        if (str.length() > width)
            return str.substr(0, width); // Truncate the string
        else
            return str; // Return the original string if its length is within the width
    }
};
// Parser is the class responsible for sorting rides based on their date and time.
// This class also manages the movement of completed rides to the past rides file, ensuring that the upcoming rides file contains only future rides.
// This segregation facilitates the implementation of features such as carpooling and ensures that users have access to relevant and timely ride information.
class Parser
{
public:
    void sortRides(string upcoming, string past)
    {
        vector<Ride> upcomingRides = parseRides(upcoming);
        vector<Ride> pastRides = parseRides(past);
        // Remove upcoming rides older than or equal to the current date and time
        time_t currentTime = time(nullptr);
        struct tm *now = localtime(&currentTime);
        int currentYear = now->tm_year + 1900;
        int currentMonth = now->tm_mon + 1;
        int currentDay = now->tm_mday;
        int currentHour = now->tm_hour;
        vector<Ride> newUpcomingRides;
        for (auto it = upcomingRides.begin(); it != upcomingRides.end();)
        {
            string rideDate = it->getDate();
            string rideTime = it->getTime();
            // Parse ride date
            int rideYear, rideMonth, rideDay;
            sscanf(rideDate.c_str(), "%d-%d-%d", &rideYear, &rideMonth, &rideDay);
            // Parse ride time
            int rideHour, rideMinute;
            sscanf(rideTime.c_str(), "%d:%d", &rideHour, &rideMinute);
            // Compare dates
            if (rideYear > currentYear || (rideYear == currentYear && rideMonth > currentMonth) || (rideYear == currentYear && rideMonth == currentMonth && rideDay > currentDay) || (rideYear == currentYear && rideMonth == currentMonth && rideDay == currentDay && rideHour > currentHour))
            {
                newUpcomingRides.push_back(*it);
                it = upcomingRides.erase(it);
            }
            else
```

```
            {
                pastRides.push_back(*it);
                it = upcomingRides.erase(it);
            }
        }
        // Sort rides based on date for upcoming and past rides
        sort(pastRides.begin(), pastRides.end(), [](Ride &a, Ride &b)
            { return a.getDate() < b.getDate() || (a.getDate() == b.getDate() && a.getTime() < b.getTime()); });
        sort(newUpcomingRides.begin(), newUpcomingRides.end(), [](Ride &a, Ride &b)
            { return a.getDate() < b.getDate() || (a.getDate() == b.getDate() && a.getTime() < b.getTime()); });
        // Write upcoming and past rides to their respective files
        writeRides(upcoming, newUpcomingRides);
        writeRides(past, pastRides);
    }
    vector<Ride> parseRides(string fileName)
    {
        vector<Ride> rides;
        ifstream file(fileName);
        string line;
        getline(file, line); // skip header
        getline(file, line); // skip ----------
        while (getline(file, line))
        {
            Ride ride(line);
            rides.push_back(ride);
        }
        file.close();
        return rides;
    }
    /**
     * Writes the details of rides to a file in a tabular format.
     * @param fileName The name of the file to write the ride details to.
     * @param rides A vector containing Ride objects with details to be written.*/
    void writeRides(string fileName, vector<Ride> &rides)
    {
        ofstream file(fileName);
        file << "| Ride ID    | Date       | Time   | Source City  | Destination City  | Max Passengers | Current
Passengers  | Car Model     | Fare(Rs) | Distance (km) |" << endl;
        file << "|------------|-------------|--------|--------------|-------------------|----------------|---------------------|-------
--------|----------|---------------|" << endl;
        for (Ride &ride : rides)
        {
            file << ride.toString() << endl;
        }
        file.close();
    }
};
class BillCalculator
{
    const double FUEL_COST_PER_KM = 12.0;
    const double MAINTENANCE_COST_PER_KM = 0.8;
    const double TOLL_COST_PER_KM = 2.0;
    // Simple distance lookup table for preset supported Indian cities
    string distanceLookup[110][3] =
        {
```

```
        {"Mumbai", "Delhi", "1200"}, {"Mumbai", "Bangalore", "980"}, {"Mumbai", "Kolkata", "1960"},
{"Mumbai", "Chennai", "1340"}, {"Mumbai", "Hyderabad", "620"}, {"Mumbai", "Pune", "150"},
{"Mumbai", "Jaipur", "1200"}, {"Mumbai", "Ahmedabad", "530"}, {"Mumbai", "Lucknow", "1500"},
{"Mumbai", "Bhopal", "770"}, {"Delhi", "Bangalore", "1800"}, {"Delhi", "Kolkata", "1470"}, {"Delhi",
"Chennai", "2180"}, {"Delhi", "Mumbai", "1200"}, {"Delhi", "Hyderabad", "1500"}, {"Delhi", "Pune",
"1550"}, {"Delhi", "Jaipur", "280"}, {"Delhi", "Ahmedabad", "940"}, {"Delhi", "Lucknow", "440"},
{"Delhi", "Bhopal", "780"}, {"Bangalore", "Mumbai", "980"}, {"Bangalore", "Delhi", "1800"},
{"Bangalore", "Kolkata", "2000"}, {"Bangalore", "Chennai", "350"}, {"Bangalore", "Hyderabad", "560"},
{"Bangalore", "Pune", "840"}, {"Bangalore", "Jaipur", "1700"}, {"Bangalore", "Ahmedabad", "1200"},
{"Bangalore", "Lucknow", "2000"}, {"Bangalore", "Bhopal", "1230"}, {"Kolkata", "Mumbai", "1960"},
{"Kolkata", "Delhi", "1470"}, {"Kolkata", "Bangalore", "2000"}, {"Kolkata", "Chennai", "1600"},
{"Kolkata", "Hyderabad", "1450"}, {"Kolkata", "Pune", "1800"}, {"Kolkata", "Jaipur", "1600"},
{"Kolkata", "Ahmedabad", "1850"}, {"Kolkata", "Lucknow", "1050"}, {"Kolkata", "Bhopal", "1450"},
{"Chennai", "Mumbai", "1340"}, {"Chennai", "Delhi", "2180"}, {"Chennai", "Bangalore", "350"},
{"Chennai", "Kolkata", "1600"}, {"Chennai", "Hyderabad", "680"}, {"Chennai", "Pune", "1200"},
{"Chennai", "Jaipur", "2300"}, {"Chennai", "Ahmedabad", "1700"}, {"Chennai", "Lucknow", "2000"},
{"Chennai", "Bhopal", "1650"}, {"Hyderabad", "Mumbai", "620"}, {"Hyderabad", "Delhi", "1500"},
{"Hyderabad", "Bangalore", "560"}, {"Hyderabad", "Kolkata", "1450"}, {"Hyderabad", "Chennai", "680"},
{"Hyderabad", "Pune", "620"}, {"Hyderabad", "Jaipur", "1100"}, {"Hyderabad", "Ahmedabad", "970"},
{"Hyderabad", "Lucknow", "1600"}, {"Hyderabad", "Bhopal", "1020"}, {"Pune", "Mumbai", "150"},
{"Pune", "Delhi", "1550"}, {"Pune", "Bangalore", "840"}, {"Pune", "Kolkata", "1800"}, {"Pune",
"Chennai", "1200"}, {"Pune", "Hyderabad", "620"}, {"Pune", "Jaipur", "1300"}, {"Pune", "Ahmedabad",
"650"}, {"Pune", "Lucknow", "1700"}, {"Pune", "Bhopal", "1030"}, {"Jaipur", "Mumbai", "1200"},
{"Jaipur", "Delhi", "280"}, {"Jaipur", "Bangalore", "1700"}, {"Jaipur", "Kolkata", "1600"}, {"Jaipur",
"Chennai", "2300"}, {"Jaipur", "Hyderabad", "1100"}, {"Jaipur", "Pune", "1300"}, {"Jaipur",
"Ahmedabad", "660"}, {"Jaipur", "Lucknow", "550"}, {"Ahmedabad", "Mumbai", "530"}, {"Ahmedabad",
"Delhi", "940"}, {"Ahmedabad", "Bangalore", "1200"}, {"Ahmedabad", "Kolkata", "1850"},
{"Ahmedabad", "Chennai", "1700"}, {"Ahmedabad", "Hyderabad", "970"}, {"Ahmedabad", "Pune",
"650"}, {"Ahmedabad", "Jaipur", "660"}, {"Ahmedabad", "Lucknow", "1350"}, {"Ahmedabad", "Bhopal",
"580"}, {"Lucknow", "Mumbai", "1500"}, {"Lucknow", "Delhi", "440"}, {"Lucknow", "Kolkata", "1050"},
{"Lucknow", "Chennai", "2000"}, {"Lucknow", "Hyderabad", "1600"}, {"Lucknow", "Pune", "1700"},
{"Lucknow", "Jaipur", "550"}, {"Lucknow", "Ahmedabad", "1350"}, {"Lucknow", "Bhopal", "650"},
{"Bhopal", "Mumbai", "770"}, {"Bhopal", "Delhi", "780"}, {"Bhopal", "Bangalore", "1230"}, {"Bhopal",
"Kolkata", "1450"}, {"Bhopal", "Chennai", "1650"}, {"Bhopal", "Hyderabad", "1020"}, {"Bhopal", "Pune",
"1030"}, {"Bhopal", "Jaipur", "580"}, {"Bhopal", "Ahmedabad", "650"}, {"Bhopal", "Lucknow", "650"}};
public:
    /**
     * Calculates the distance between a given source and destination.
     * @param source The source location.
     * @param destination The destination location.
     * @returns The distance between the source and destination.*/
    int calculateDistance(string source, string destination)
    {
        for (int i = 0; i < 110; i++)
        {
            if (distanceLookup[i][0] == source && distanceLookup[i][1] == destination)
            {
                return stoi(distanceLookup[i][2]);
            }
        }
        int dis;
        cout << "Enter the distance between source and destination : ";
        cin >> dis;
        return dis;
    }
```

```
// Function to calculate the total cost based on distance, car capacity, and fixed costs
/**
 * Calculates the total cost of a trip based on distance and car capacity.
 * @param distance The total distance of the trip.
 * @param carCapacity The capacity of the car (number of passengers).
 * @return The total cost of the trip, including fuel, maintenance, tolls, and additional costs.*/
double calculateTotalCost(int distance, int carCapacity)
{
    double fuelCost = distance * FUEL_COST_PER_KM;
    double maintenanceCost = distance * MAINTENANCE_COST_PER_KM;
    double tollCost = distance * TOLL_COST_PER_KM;
    double additionalCost = carCapacity * 20.0; // Adjust this value as needed
    double totalCost = fuelCost + maintenanceCost + tollCost + additionalCost;
    return totalCost;
}
// Function to calculate reward points based on distance
/**
 * Calculates the points earned based on the distance travelled */
int calculatePoints(int distance)
{
    return (distance / 100) * 5; // 10 points per 100 km
}
// Function to determine incentives based on points and user ID
/**
 * Determines the incentive based on the points earned and saves the details to a file.
 * @returns The incentive message based on the points earned.*/
string determineIncentive(int points, string z)
{
    string incentive;
    if (points >= 100 && points < 200)
    {
        incentive = "Discount coupon on the next ride";
    }
    else if (points >= 200 && points < 300)
    {
        incentive = "Free upgrade to premium vehicle on the next ride";
    }
    else if (points >= 300)
    {
        incentive = "Free ride up to a certain distance";
    }
    else
    {
        incentive = "Free spotify premium";
    }
    // Save incentive details based on user ID
    if (!incentive.empty())
    {
        ofstream incentiveFile("CodeRelatedFiles/rewardstaken.txt", ios::app);
        if (incentiveFile.is_open())
        {
            incentiveFile << "Username: " << z << " Incentive " << incentive << endl;
            incentiveFile.close();
        }
    }
```

```
        return incentive;
    }
    // Function to print the bill in a proper format
    /**
     * Generates and prints a bill for a given trip.
     * @param source The source location of the trip.
     * @param destination The destination location of the trip.
     * @param distance The distance of the trip.
     * @param carCapacity The capacity of the car for the trip.*/
    void printBill(string source, string destination, int distance, int carCapacity)
    {
        system("cls");
        system("color A1");
        double totalCost = calculateTotalCost(distance, carCapacity);
        cout << "\n";
        cout << "\n";
        cout << "\t\t\t\t\t\t"<< "\n";
        cout << "\t\t\t\t\t\t" << setw(60) << "|************************* Trip Bill
*************************|\n";
        cout << "\t\t\t\t\t\t"<< "|                                                  |" << endl;
        cout << "\t\t\t\t\t\t"<< "|\tSource City: \t\t" << setw(25) << left << source << "       |" << endl;
        cout << "\t\t\t\t\t\t"<< "|\tDestination City: \t" << setw(25) << left << destination << "       |" << endl;
        cout << "\t\t\t\t\t\t"<< "|\tDistance: \t\t" << setw(15) << left << distance << " km             |\n";
        cout << "\t\t\t\t\t\t"<< "|\tCar Capacity: \t\t" << setw(15) << left << carCapacity << " passengers     |\n";
        cout << "\t\t\t\t\t\t"<< "|\t-----------------------------------------------------|\n";
        cout << "\t\t\t\t\t\t"<< "|\tFuel Cost: \t\t\tRs." << setw(20) << left << distance * FUEL_COST_PER_KM << "|"
<< endl;
        cout << "\t\t\t\t\t\t"<< "|\tMaintenance Cost: \t\tRs." << setw(20) << left << distance *
MAINTENANCE_COST_PER_KM << "|" << endl;
        cout << "\t\t\t\t\t\t"<< "|\tToll Cost: \t\t\tRs." << setw(20) << left << distance * TOLL_COST_PER_KM << "|"
<< endl;
        cout << "\t\t\t\t\t\t"<< "|\tAdditional Cost: \t\tRs." << setw(20) << left << carCapacity * 20.0 << "|" << endl;
        cout << "\t\t\t\t\t\t"<< "|\tDriver Cost: \t\t\tRs." << setw(20) << left << 200 << "|" << endl;
        cout << "\t\t\t\t\t\t"<< "|\t-----------------------------------------------------|\n";
        cout << "\t\t\t\t\t\t"<< "|\tTotal Cost: \t\t\tRs." << setw(10) << left << totalCost + 200 << "   only    |" << endl;
        cout << "\t\t\t\t\t\t"<< "|                                                  |" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|   # This is a computer-generated invoice and it does not     |" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|     require an authorized signature #                    |" << endl;
        cout << "\t\t\t\t\t\t"<< "|                                                  |" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|//////////////////////////////////////////////////////////|" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|You are advised to pay up the amount before the due date.     |" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|\t\tOtherwise, a penalty fee will be applied       |" << endl;
        cout << "\t\t\t\t\t\t"<< setw(60) << "|//////////////////////////////////////////////////////////|" << endl;
        cout << "\t\t\t\t\t\t"<< "|                                                  |" << endl;
        cout << "\t\t\t\t\t\t" << setw(60) << "************************* Thank You! *************************\n";
    }
};
/**
 * Class representing a menu with options for users.
 * Extends Pages and BillCalculator classes.
 */
class Menu : public Pages, public BillCalculator
{
public:
    // Display the menu options
    /**
     * Displays the menu options for the user to interact with the ride-sharing system.
     * @param user A pointer to the User object for whom the menu is being displayed.*/
    void displayMenu(User *user)
```

```
    {
        // this is just a outline you will have to modify it for proper working
        cout << endl;
        cout << "\t\t\t\t\t||========================================== Menu
==========================================||" << endl;
        cout << "\t\t\t\t\t|| 1. Book a Ride\t\t2. Offer a Ride\t\t3. View Upcoming Rides \t\t  ||" << endl;
        cout << "\t\t\t\t\t|| 4. View Past Rides\t\t5. Manage Profile\t6. Print Bills     \t\t  ||" << endl;
        cout << "\t\t\t\t\t|| 7. Search for Rides\t\t8. Join a Pool\t\t9. Cancel Booking  \t\t  ||" << endl;
        cout << "\t\t\t\t\t|| 10. Rate and Review\t\t11. Feedback\t\t12. Invite Friends   \t\t  ||" << endl;
        cout << "\t\t\t\t\t|| 13. View Rewards/Incentives\t14. CO2 Calculator\t15. Exit \t\t\t  ||" << endl;
        cout <<
"\t\t\t\t\t||=========================================================================================
==========||" << endl;
        bool flag = false;
        do
        {
            int choice;
            cout << "Enter your choice: ";
            cin >> choice;
            // Execute the selected option based on user input
            switch (choice)
            {
            case 1:
                flag = true;
                bookRide(user);
                break;
            case 2:
                flag = true;
                offerRide();
                break;
            case 3:
                flag = true;
                viewUpcomingRides(user);
                break;
            case 4:
                flag = true;
                viewPastRides(user);
                break;
            case 5:
                flag = true;
                manageProfile(user);
                break;
            case 6:
                flag = true;
                printBills(user);
                break;
            case 7:
                flag = true;
                searchRides();
                break;
            case 8:
                flag = true;
                joinPool();
                break;
            case 9:
                flag = true;
```

```
            cancelBooking();
            break;
        case 10:
            flag = true;
            rateAndReview();
            break;
        case 11:
            flag = true;
            feedback();
            break;
        case 12:
            flag = true;
            inviteFriends();
            break;
        case 13:
            flag = true;
            viewRewards();
            break;
        case 14:
            flag = true;
            calculateCO2Emission();
            break;
        case 15:
            flag = true;
            exitProgram();
            break;
        default:
            cout << "Invalid choice. Please enter a number between 1 and 15" << endl;
            flag = false;
        }
    } while (flag == false);
}
private:
/**
 * Books a ride for the given user.
 * This function initiates the process of booking a ride for the user by creating a new Ride object
 * with the provided details such as date, time, source city, destination city, car model, maximum
passengers,
 * current passengers, fare, and distance. The ride information is then stored in a file for both the user
 * and the admin to keep track of upcoming rides.
 * @param user A pointer to the User object for whom the ride is being booked.*/
void bookRide(User *user)
{
    system("color E1");
    system("cls");

    cout << "Booking a ride..." << endl;
    // Implementing logic for booking a ride
    string rideID, date, time, sourceCity, destinationCity, carModel;
    int maxPassengers, currentPassengers;
    double fare, distance;
    cout << "Enter Date (YYYY-MM-DD): ";
    cin >> date;
    cout << "Enter Time (HH:MM): ";
    cin >> time;
```

```
        cout << "Enter Source City: ";
        cin >> sourceCity;
        cout << "Enter Destination City: ";
        cin >> destinationCity;
        cout << "Enter Max Passengers: ";
       cin >> maxPassengers;
       cout << "Enter Current Passengers: ";
       cin >> currentPassengers;
       cout << "Enter Car Model: ";
       cin >> carModel;
       distance = calculateDistance(sourceCity, destinationCity);
       fare = calculateTotalCost(distance, maxPassengers);
       Ride r(date, time, sourceCity, destinationCity, maxPassengers, currentPassengers, carModel, fare, distance);
        string s = r.toString();
        string fileName = "./Files/" + user->getUsername() + "/upcomingRides.txt";
        string adminName = "./Files/admin/upcomingRides.txt";
        ofstream file(fileName, ios::app);    // Open file in append mode
        ofstream adfile(adminName, ios::app); // Open file in append mode
        file << s << endl;
        adfile << s << endl;
    }
    /**
     * Offers a ride by taking input from the user and creating a new Ride object.*/
    void offerRide()
    {
        system("color 4F");
        system("cls");
        cout << "Offering a ride..." << endl;
        string date, time, sourceCity, destinationCity, carModel;
        int maxPassengers, currentPassengers;
        double fare, distance;
        cout << "Enter date (DD/MM/YYYY): ";
        cin >> date;
        cout << "Enter time (HH:MM): ";
        cin >> time;
        cout << "Enter source city: ";
        cin >> sourceCity;
        cout << "Enter destination city: ";
        cin >> destinationCity;
        cout << "Enter maximum passengers: ";
        cin >> maxPassengers;
        cout << "Enter current passengers: ";
        cin >> currentPassengers;
        cout << "Enter car model: ";
        cin >> carModel;
        distance = calculateDistance(sourceCity, destinationCity);
        fare = calculateTotalCost(distance, maxPassengers);
        Ride newRide(date, time, sourceCity, destinationCity, maxPassengers, currentPassengers, carModel,
fare, distance);
        cout << "Ride offered successfully!" << endl;
    }
    /**
     * Displays the upcoming rides for a given user.
     * This function clears the console screen, sets the text color, and reads and displays the upcoming rides
     * from the user's file.
     * @param user A pointer to the User object for whom the upcoming rides are to be displayed.*/
```

```
void viewUpcomingRides(User *user)
{
    system("color 1F");
    system("cls");
    cout << "Viewing upcoming rides..." << endl;
    string fileName = "./Files/" + user->getUsername() + "/upcomingRides.txt";
    vector<Ride> rides;
    ifstream file(fileName);
    if (!file.is_open())
    {
        cout << "Error: Could not open file." << endl;
        return;
    }
    string line;
    cout << "-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
    while (getline(file, line))
    {
        cout << line << endl;
    }
    cout << "-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
    cout << endl
         << endl;
}
/**
 * Displays the past rides of a user by reading from a file.
 * @param user A pointer to the User object whose past rides are to be viewed.*/
void viewPastRides(User *user)
{
    system("color 5A");
    system("cls");
    cout << "Viewing past rides..." << endl;
    string fileName = "./Files/" + user->getUsername() + "/pastRides.txt";
    vector<Ride> rides;
    ifstream file(fileName);
    if (!file.is_open())
    {
        cout << "Error: Could not open file." << endl;
        return;
    }
    string line;
    cout << "-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
    while (getline(file, line))
    {
        cout << line << endl;
    }
    cout << "-------------------------------------------------------------------------------------------------------------------------------------------" << endl;
    cout << endl
         << endl;
}
/**
 * Manages the user profile by updating user details in a file.
```

```
 * @param user Pointer to the User object whose profile is being managed.*/
void manageProfile(User *user)
{
    system("color 3F");
    system("cls");
    cout << "Managing profile..." << endl;
    string fileName = "./Files/" + user->getUsername() + "/userDetails.txt";
    ofstream file(fileName);
    string username, fullName, age, gender, email, address, phone, aadharNo, memberSince, ridesTaken,
amountSpent;
    username = fileName;
    cout << "Enter Full Name: ";
    getline(cin, fullName);
    getline(cin, fullName);
    cout << "Enter Age: ";
    getline(cin, age);
    cout << "Enter Gender: ";
    getline(cin, gender);
    cout << "Enter Email: ";
    getline(cin, email);
    cout << "Enter Address: ";
    getline(cin, address);
    cout << "Enter Phone: ";
    getline(cin, phone);
    cout << "Enter Aadhar Card No: ";
    getline(cin, aadharNo);
    // Write user details to file
    file << "Username: " << username << endl;
    file << "Full Name: " << fullName << endl;
    file << "Age: " << age << endl;
    file << "Gender: " << gender << endl;
    file << "Email: " << email << endl;
    file << "Address: " << address << endl;
    file << "Phone: " << phone << endl;
    file << "Aadhar Card No: " << aadharNo << endl;
    file << "Member Since: " << memberSince << endl;
    string fileName1 = "./Files/" + user->getUsername() + "/pastRides.txt";
    string fileName2 = "./Files/" + user->getUsername() + "/upcomingRides.txt";
    ifstream file1(fileName1);
    ifstream file2(fileName2);
    int totalLines = 0;
    string line;
    while (getline(file1, line))
    {
        totalLines++;
    }
    while (getline(file2, line))
    {
        totalLines++;
    }
    file1.close();
    file2.close();
    file << "Total Rides Taken: " << totalLines - 4 << endl;
    // Close the file
    file.close();
```

```
    }
    /**
     * Prints the bill details for a specific ride of a user.
     * @param user A pointer to the User object for whom the bill is to be printed.*/
    void printBills(User *user)
    {
        system("cls");
        string id;
        cout << "Please enter Ride id:" << endl;
        cin >> id;
        string fileName = "./Files/" + user->getUsername() + "/pastRides.txt";
        string upcomingFileName = "./Files/" + user->getUsername() + "/upcomingRides.txt";
        vector<Ride> rides;
        ifstream file(fileName);
        string line;
        getline(file, line); // skip header
        getline(file, line); // skip ----------

        while (getline(file, line))
        {
            Ride ride(line);
            rides.push_back(ride);
        }
        file.close();
        bool found = false;
        for (Ride &ride : rides)
        {
            if (ride.getRideID().substr(0, 5) == id)
            {
                found = true;
                Sleep(500);
                // Assuming you have a function to retrieve bill information from the Ride object
                cout << "Bill details for ride with ID " << id << ":" << endl;
                BillCalculator b;
                b.printBill(ride.getSourceCity(), ride.getDestinationCity(), ride.getDistance(),
ride.getMaxPassengers());
                break; // No need to continue searching once found
            }
        }
        if (!found)
        {
            cout << "Ride with ID " << id << " not found in past rides." << endl;
            vector<Ride> upcomingRides;
            ifstream upcomingFile(upcomingFileName);
            getline(upcomingFile, line); // skip header
            getline(upcomingFile, line); // skip header
            while (getline(upcomingFile, line))
            {
                Ride ride(line);
                upcomingRides.push_back(ride);
            }
            upcomingFile.close();
            for (Ride &ride : upcomingRides)
            {
                if (ride.getRideID().substr(0, 5) == id)
```

```
            {
                found = true;
                Sleep(500);
                // Assuming you have a function to retrieve bill information from the Ride object
                cout << "Bill details for upcoming ride with ID " << id << ":" << endl;
                BillCalculator b;
                b.printBill(ride.getSourceCity(), ride.getDestinationCity(), ride.getDistance(),
ride.getMaxPassengers());
                break; // No need to continue searching once found
            }
        }
    }
    if (!found)
    {
        cout << "Ride with ID " << id << " was not found." << endl;
    }
}
/**
 * Searches for available rides based on user input for source city and destination city.
 * If rides are found, displays the details of the available rides.*/
void searchRides()
{
    system("cls");
    cout << "Searching for rides..." << endl;
    string sourceCity, destinationCity, date, time;
    cout << "Enter source city: ";
    cin >> sourceCity;
    cout << "Enter destination city: ";
    cin >> destinationCity;
    string adminName = "./Files/admin/upcomingRides.txt";
    string line;
    ifstream file(adminName);
    if (!file.is_open())
    {
        cout << "Error: Could not open file." << endl;
        return;
    }
    vector<Ride> rides;
    bool found = false;
    getline(file, line); // skip header
    getline(file, line); // skip ----------
    string rided = "";
    while (getline(file, line))
    {
        Ride ride(line);
        rides.push_back(ride);
        if (ride.getSourceCity() == sourceCity && ride.getDestinationCity() == destinationCity)
        {
            found = true;
            rided = rided + ride.toString() + "\n";
        }
    }
    if (!found)
    {
        cout << "No matching ride found." << endl;
```

```
        }
        else
        {
            cout << "Rides were found.....";
            cout << "| Ride ID    | Date       | Time   | Source City  | Destination City  | Max Passengers | Current
Passengers  | Car Model     | Fare(Rs) | Distance (km) |" << endl;
            cout << "|------------|------------|--------|--------------|-------------------|----------------|--------------------|--
-----------|----------|---------------|" << endl;
            cout << rided;
        }
        file.close();
    }
    /**
     * Allows a user to join a carpool pool by entering the number of people.
     * If the input is invalid or no available rides match the given source and destination,
     * or all available seats are already booked, appropriate messages are displayed.*/
    void joinPool()
    {
        cout << "Joining a pool..." << endl;
        int numPeople;
        cout << "Enter the number of people for carpool: ";
        cin >> numPeople;
        if (cin.fail() || numPeople <= 0)
        {
            cout << "Invalid input for number of people." << endl;
            return;
        }
        cin.ignore();
        string sourceCity, destinationCity;
        cout << "Enter Source City: ";
        getline(cin, sourceCity);
        cout << "Enter Destination City: ";
        getline(cin, destinationCity);
        ifstream file("Files/admin/upcomingRides.txt");
        ofstream tempFile("Files/admin/temp.txt");
        string line;
        bool rideFound = false;
        getline(file, line);
        tempFile << line << endl;
        getline(file, line);
        bool found = false;
        tempFile << line << endl;
        while (getline(file, line))
        {
            Ride ride(line);
            if (found == false && ride.getSourceCity() == sourceCity && ride.getDestinationCity() ==
destinationCity && ride.getCurrentPassengers() < ride.getMaxPassengers())
            {
              rideFound = true;
              cout << "Ride details:" << endl;
              cout << "Ride ID: " << ride.getRideID() << endl;
              cout << "Date: " << ride.getDate() << endl;
              cout << "Time: " << ride.getTime() << endl;
              cout << "Car Model: " << ride.getCarModel() << endl;
             cout << "Fare: " << fixed << setprecision(2) << ride.getFare() * 0.9 << " Rs" << endl; // Reduced fare by 10%
              cout << "Distance: " << ride.getDistance() << " km" << endl;
```

```
            string choice;
            cout << "Do you want to take this ride? (y(yes)/n(no)): ";
            cin >> choice;
            if (choice == "y" || choice == "yes")
             {
                if (ride.getCurrentPassengers() < ride.getMaxPassengers())
                {
                    ride.setCurrentPassengers(ride.getCurrentPassengers() + numPeople);
                    cout << "Ride booked successfully!" << endl;
                    found = true;
                }
                else
                {
                    cout << "No seats available for this ride." << endl;
                }
             }
          }
          tempFile << ride.toString() << endl;
       }
       file.close();
       tempFile.close();
       if (!rideFound)
       {
          cout << "No available rides match the given source and destination, or all available seats are already
booked." << endl;
       }
       remove("Files/admin/upcomingRides.txt");
       rename("Files/admin/temp.txt", "Files/admin/upcomingRides.txt");
    }
    /**
     * Cancels a booking based on user's choice of cancellation method.
     * The function prompts the user to choose between canceling by date or time.
     * After the user makes a choice, the booking is canceled accordingly.
     * The function then outputs a success message and closes the output file.*/
    void cancelBooking()
    {
       int choice;
       cout << "How would you like to cancel your booking?" << endl;
       cout << "1. By Date" << endl;
       cout << "2. By Time" << endl;
       cout << "Enter your choice: ";
       cin >> choice;
       string cancelDate, cancelTime;
       switch (choice)
       {
       case 1:
          cout << "Enter the Date (YYYY-MM-DD) of the booking to cancel: ";
          cin >> cancelDate;
          break;
       case 2:
          cout << "Enter the Time (HH:MM) of the booking to cancel: ";
          cin >> cancelTime;
          break;
       default:
          cout << "Invalid choice. Please enter 1 or 2." << endl;
          return;
```

```
    }
    ifstream inFile("Files/admin/upcomingRides.txt");
    if (!inFile)
    {
        cerr << "Error: Unable to open file." << endl;
        return;
    }
    string line;
    vector<string> modifiedLines;
    bool found = false;
    while (getline(inFile, line))
    {
        // Check if the line contains the entered Date or Time
        if ((choice == 1 && line.find(cancelDate) != string::npos) ||
            (choice == 2 && line.find(cancelTime) != string::npos))
        {
            found = true;
            continue;
        }
        // Add the line to the modifiedLines vector
        modifiedLines.push_back(line);
    }
    inFile.close();
    if (!found)
    {
        if (choice == 1)
        {
            cout << "Booking with Date " << cancelDate << " not found. No booking canceled." << endl;
        }
        else
        {
            cout << "Booking with Time " << cancelTime << " not found. No booking canceled." << endl;
        }
        return;
    }
    // Rewriting the modified contents back to the file
    ofstream outFile("Files/admin/upcomingRides.txt", ofstream::out | ofstream::trunc);
    if (!outFile)
    {
        cerr << "Error: Unable to open file for writing." << endl;
        return;
    }
    for (const string &l : modifiedLines)
    {
        outFile << l << endl;
    }
    cout << "Booking canceled successfully." << endl;
    outFile.close();
}
/**
 * Displays a rating prompt for the user to rate a service or product.
 * The user can choose a rating from 1 to 5 stars.*/
void rateAndReview()
{
    cout << "\nRATE US NOW!!" << endl;
```

```
    cout << "1. *\n";
    cout << "2. **\n";
    cout << "3. ***\n";
    cout << "4. ****\n";
    cout << "5. *****\n";
    int rate;
    cin >> rate;

    // Clear input buffer to avoid potential issues
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
/**
 * Collects feedback from the user and saves it to a file.
 * This function prompts the user to provide feedback, reads the input,
 * and appends it to a file named 'feedback.txt' in the 'CodeRelatedFiles' directory.
 * If the file is successfully opened, the feedback is saved, and a confirmation message is displayed.
 * If there is an error opening the file, an error message is shown.*/
void feedback()
{
    cout << "Providing feedback..." << endl;
    string feedback;
    cout << "Please provide your feedback (up to 999 characters):\n";
    getline(cin >> ws, feedback);
    // Saving feedback to a file
    ofstream feedbackFile("CodeRelatedFiles/feedback.txt", ios::app);
    if (feedbackFile.is_open())
    {
        feedbackFile << feedback << endl;
        feedbackFile.close();
        cout << "Thank you for your feedback! It has been saved in feedback.txt." << endl;
    }
    else
    {
        cout << "Error saving feedback. Please try again later." << endl;
    }
}
/**
 * Allows the user to invite friends to an upcoming ride by providing the ride ID.
 * Reads the upcoming rides information from a file and checks if the provided ride ID exists.
 * If the ride ID is found, the user can proceed with inviting friends.
 * If the ride ID is not found, a message is displayed indicating that no ride was found with the given ID.*/
void inviteFriends()
{
    ifstream file("Files/admin/upcomingRides.txt");
    string line;
    bool rideFound = false;
    cout << "Enter Ride ID: ";
    string rideID;
    cin >> rideID;
    getline(file, line); // Skipping headers to avoid any errors
    getline(file, line);
    while (getline(file, line))
    {
        Ride ride(line);
       if (ride.getRideID() == rideID)
```

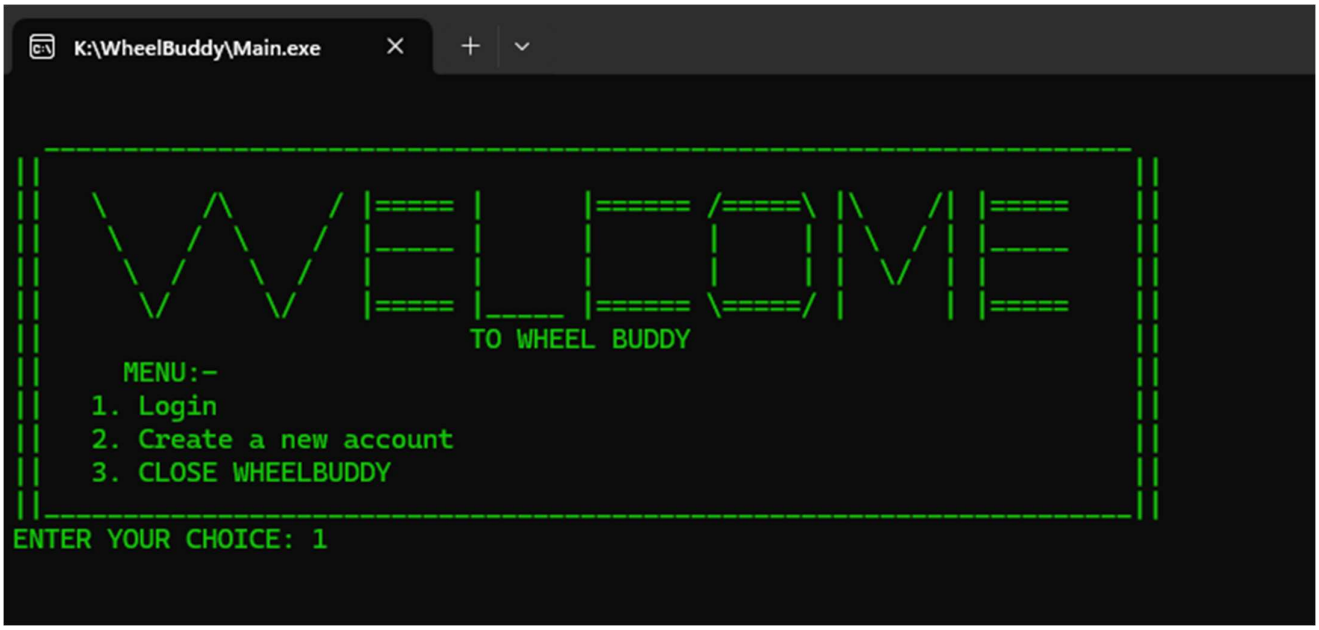
```
        {
            rideFound = true;
            cout << "Ride found!" << endl;
            cout << "Source City: " << ride.getSourceCity() << endl;
            cout << "Destination City: " << ride.getDestinationCity() << endl;
            cout << "Date: " << ride.getDate() << endl;
            cout << "Time: " << ride.getTime() << endl;
            cout << "Car Model: " << ride.getCarModel() << endl;
            cout << "Fare: " << fixed << setprecision(2) << ride.getFare() << " Rs" << endl;
            cout << "Distance: " << ride.getDistance() << " km" << endl;
            cout << "Current Passengers: " << ride.getCurrentPassengers() << endl;
            cout << "Max Passengers: " << ride.getMaxPassengers() << endl;
            if (ride.getCurrentPassengers() < ride.getMaxPassengers())
            {
                cout << "Would you like to invite a friend to this ride? (y/n): ";
                char inviteChoice;
                cin >> inviteChoice;
                if (inviteChoice == 'y' || inviteChoice == 'Y')
                {
                    // Increment the passenger count
                    ride.setCurrentPassengers(ride.getCurrentPassengers() + 1);
                    cout << "Passenger count updated." << endl;
                }
                else
                {
                    cout << "Invite cancelled." << endl;
                }
            }
            else
            {
                cout << "Sorry, this ride is at full capacity." << endl;
            }
            break;
        }
    }
    if (!rideFound)
    {
        cout << "No ride found with the given Ride ID." << endl;
    }
    file.close();
}
/*Displays the rewards earned by a user based on past rides.*/
void viewRewards()
{
    BillCalculator bill;
    string userID;
    cout << "Enter User ID: ";
    cin >> userID;
    // File path for user's past rides
    string fileName = "./Files/" + userID + "/pastRides.txt";
    ifstream inFile(fileName);
    // Skip the header line
    string header;
    getline(inFile, header);
    getline(inFile, header);
    int totalDistance = 0;
    int total_points;
```

```
        string rideInfo;
        string incentive;
        while (getline(inFile, rideInfo))
        {
            Ride ride(rideInfo);
            // Calculate points based on distance
            int points = bill.calculatePoints(ride.getDistance());
            total_points += points;
            // Determine incentives based on points and save incentive details
            incentive = bill.determineIncentive(points, userID);
            cout << "Ride ID: " << ride.getRideID() << ", Distance: " << ride.getDistance() << " km, Points: "
<< points << endl;
            // Update total distance
            totalDistance += ride.getDistance();
        }
        inFile.close();
        cout << "Total distance: " << totalDistance << " km" << endl;
        cout << "Total Points : " << total_points << endl;
        cout << "Incentive : " << incentive << endl;
    }
void calculateCO2Emission()
    {
        cout << "Calculating CO2 emission..." << endl;
        string source, destination;
        cout << "Enter source city: ";
        cin >> source;
        cout << "Enter destination city: ";
        cin >> destination;
        int distance = calculateDistance(source, destination);
        double emission = distance * 16.0; // Emission in grams
        cout << "Total CO2 emission for the trip: " << emission << " grams" << endl;
        if (emission > 500)
        {
            cout << "Your emissions are high. Consider the following measures to reduce them:" << endl;
            cout << "- Use public transportation where possible." << endl;
            cout << "- Carpool with others to reduce the number of vehicles on the road." << endl;
            cout << "- Choose fuel-efficient vehicles or consider electric options." << endl;
            cout << "- Plan your trips efficiently to minimize distance traveled." << endl;
        }
        else
        {
            cout << "Remember, as Antoine de Saint-Exupery said," << endl;
            cout << "\"We do not inherit the earth from our ancestors, we borrow it from our children.\"" <<
endl;
            cout << "Let's make choices that ensure a brighter future for the next generations." << endl;
            cout << endl;
        }
        Sleep(1000);
    }
    /**
     * Exits the program and displays a thank you message.*/
    void exitProgram()
    {
        cout << "Exiting program..." << endl;
        thankYouPage();
```

```
    }
};
/**
 * Main function to run the program.
 * This function initializes the necessary variables and objects, loads a file, displays a home page,
 * creates a menu, and continuously displays the menu until the program is exited.
 * After exiting the loop, a thank you page is displayed before the program ends.
 * @return 0 upon successful completion of the program.
 */
int main()
{
    Pages page;
    bool loggedIn = false;
    bool exitProgramFlag = false; // Flag to indicate if the program should exit
    page.fileLoadingPage();
    User *user;
    string filename = "CodeRelatedFiles/Credentials.txt";
    int choice = page.homePage();
    do
    {
        switch (choice)
        {
        case 1:
        {
            LoginManager login(filename);
            user = login.loginScreen();
            if (user->getUsername() != "null" && user->getPassword() != "null")
            {
                cout << "*************************************************\n";
                cout << "*    Login Successful, welcome to WheelBuddy    *\n";
                cout << "*************************************************\n";
                loggedIn = true;
                Sleep(3000);
                system("cls");
            }
            else
            {
                cout << "Login failed. Incorrect username or password.\n";
                cout << "Do you want to create a new account? (yes/no): ";
                string response;
                cin >> response;
                if (response == "yes")
                {
                    RegistrationManager::registerUser(filename);
                    choice = 1;
                }
                else
                {
                    choice = page.homePage();
                }
            }
            break;
        }
        case 2:
            RegistrationManager::registerUser(filename);
```

```
            choice = 1;
            exitProgramFlag = false;
            break;
        case 3:
            exitProgramFlag = true;
            break;
        default:
            cout << "Program Terminated due to wrong input";
        }
        if (exitProgramFlag)
        {
            page.thankYouPage();
            break; // Break out of the loop after calling exitProgram()
        }
    } while (!loggedIn);
    if (loggedIn == true)
    {
        // Parsing is the class for sorting the rides based on their time and date
        // Also this class moves the finished rides to a different rides so upcoming rides has only coming rides
data which can be further used to implement the features of carpooling
        // after logging in run the parser first
        string folderLocName = "./Files/" + user->getUsername(); // Replace with actual folder name
        Parser parser;
        parser.sortRides(folderLocName + "/upcomingRides.txt", folderLocName + "/pastRides.txt");
        // Also run the parser for the admin
        folderLocName = "./Files/admin";
        parser.sortRides(folderLocName + "/upcomingRides.txt", folderLocName + "/pastRides.txt");
        // Now display the dashboard
        Dashboard dash;
        dash.display(user->getUsername());
        Sleep (1000);
        Menu menu;
        // Display the menu
        while (true)
        {
            menu.displayMenu(user);
        }`
    }
    page.thankYouPage();
    return 0;
}
```

# OUTPUT SCREENS


```
************************* User Dashboard *************************
User Details:
        Username: ./Files/1234/userDetails.txt
        Full Name: Rohan
        Age: 20
        Gender: Male
        Email: sfknglnkjg
        Address: kefnelwkejjkw
        Phone: 6419366316
        Aadhar Card No: 7734166481366
        Member Since: 374824
        Total Rides Taken: 15
        Total Amount Spent: Rs. 0
-----------------------------------------------------------------

Past Rides Details:
```

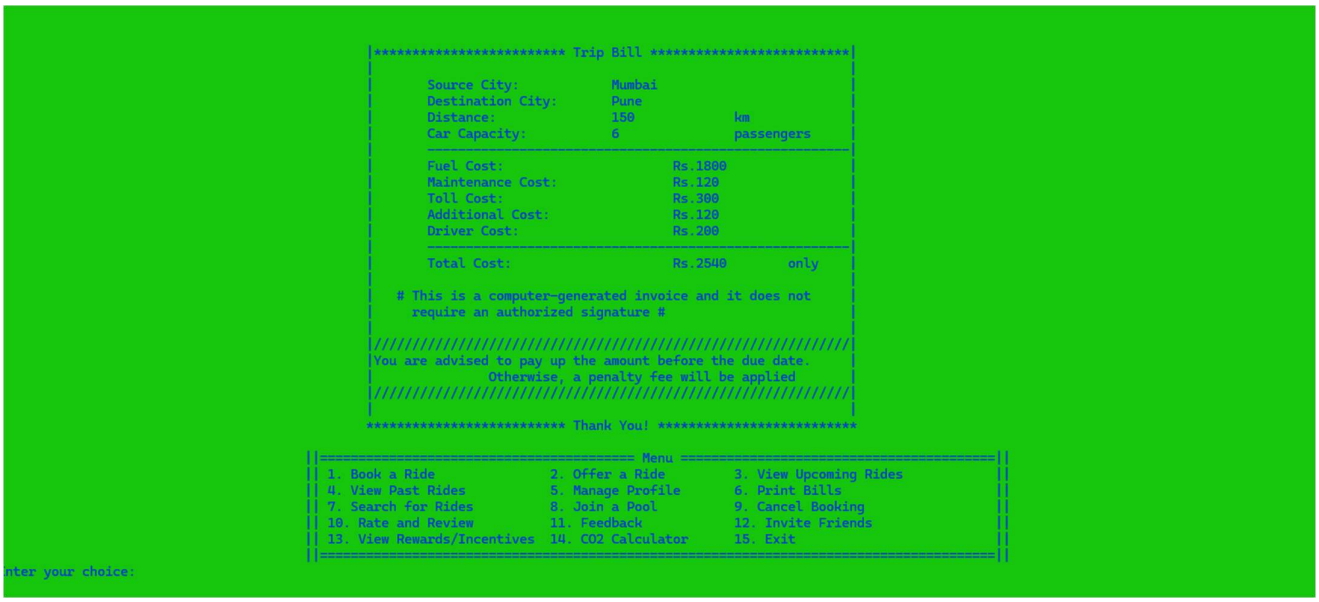
| Ride ID | Date | Time | Source City | Destination City | Max Passengers | Current Passengers | Car Model | Fare(Rs) | Distance (km) |
|---|---|---|---|---|---|---|---|---|---|
| sHLLq | 1342-23-34 | 23:23 | Delhi | Lucknow | 4 | 3 | Toyata | 6592 | 440 |
| lqace | 2002-05-10 | 12:34 | Mumbai | Pune | 7 | 2 | Fortuner | 2360 | 150 |
| S82fL | 2022-10-08 | 04:07 | delhi | noida | 5 | 4 | hjfjh | 7500 | 500 |
| A1B2C | 2023-12-20 | 08:00 | Mumbai | Delhi | 4 | 3 | Toyota Camry | 1200 | 1200 |
| D3E4F | 2023-12-25 | 12:30 | Mumbai | Bangalore | 4 | 2 | Honda Civic | 980 | 980 |
| G5H6I | 2024-01-05 | 09:15 | Mumbai | Kolkata | 6 | 5 | Maruti Swift | 1960 | 1960 |
| J7K8L | 2024-01-10 | 19:00 | Mumbai | Chennai | 5 | 4 | Hyundai i20 | 1340 | 1340 |
| M9N0O | 2024-02-02 | 16:45 | Mumbai | Hyderabad | 3 | 3 | Ford Figo | 620 | 620 |
| P1Q2R | 2024-03-30 | 10:30 | Mumbai | Pune | 4 | 0 | Toyota Innova | 150 | 150 |
| V5W6X | 2024-04-10 | 14:00 | Mumbai | Ahmedabad | 4 | 3 | Tata Nexon | 530 | 530 |

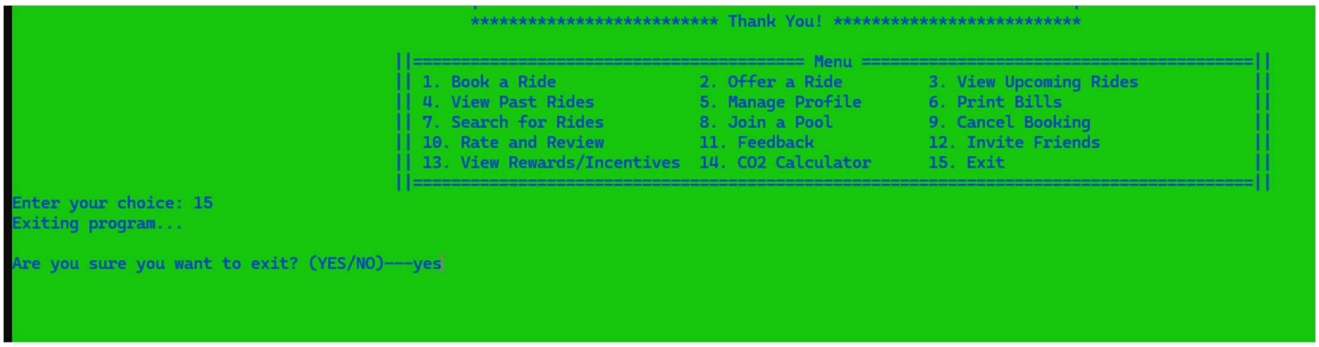
```
Upcoming Rides Details:
```

| Ride ID | Date | Time | Source City | Destination City | Max Passengers | Current Passengers | Car Model | Fare(Rs) | Distance (km) |
|---|---|---|---|---|---|---|---|---|---|
| B1C2D | 2024-04-25 | 16:20 | Mumbai | Bhopal | 3 | 2 | Renault Kwid | 770 | 770 |
| hvrzA | 2024-04-30 | 12:45 | Delhi | Lucknow | 5 | 1 | creta | 6612 | 440 |
| S3T4U | 2024-06-05 | 08:45 | Mumbai | Jaipur | 5 | 1 | Mahindra XUV | 1200 | 1200 |
| Y7Z8A | 2024-07-20 | 11:15 | Mumbai | Lucknow | 6 | 4 | Volkswagen | 1500 | 1500 |
| fCjiq | 2024-6-15 | 12:34 | Delhi | Pune | 4 | 5 | toyata | 23020 | 1550 |

```
                                ||====================================== Menu ======================================||
                                || 1. Book a Ride               2. Offer a Ride         3. View Upcoming Rides  ||
                                || 4. View Past Rides           5. Manage Profile       6. Print Bills          ||
                                || 7. Search for Rides          8. Join a Pool          9. Cancel Booking       ||
                                || 10. Rate and Review          11. Feedback            12. Invite Friends      ||
                                || 13. View Rewards/Incentives  14. CO2 Calculator      15. Exit                ||
                                ||==================================================================================||
Enter your choice:
```

```
Booking a ride...
Enter Date (YYYY-MM-DD): 2024-04-27
Enter Time (HH:MM): 12:45
Enter Source City: Mumbai
Enter Destination City: Delhi
Enter Max Passengers: 5
Enter Current Passengers: 2
Enter Car Model: Sedan
```




CONCLUSION-: **Wheels Buddy offers a comprehensive solution for modern commuting challenges by combining carpooling with detailed expense tracking. It facilitates community-building and eco-consciousness through features like user profiles, route planning, expense sharing, and secure verification. By providing a platform for seamless carpooling and reliable expense management, Wheels Buddy aims to make daily commutes efficient, convenient, and environmentally friendly.**