

TRIPMATE – CARPOOLING WEB APPLICATION

PROJECT REPORT

1. Cover Page

Project Title : **TripMate – Carpooling Web Application**
Course : **Environmental Values (EVS) Project**
Submitted By : **MALEPATTU KARTHIK**
Roll Number : **25ME10096**
Department : **INTEGRATED MTECH IN CYBER SECURITY**
Submitted To : **Pavithra R R mam**
Institution : **VIT BHOPAL UNIVERSITY**
Academic Year : **2025–2026**

2. Introduction

TripMate is a simple and user-friendly carpooling platform designed to reduce travel costs, traffic congestion, and carbon emissions. The application allows users to offer rides, find rides, and travel together, making transportation more economical and eco-friendly.

This project aims to demonstrate how technology can contribute to environmental sustainability and promote resource sharing.

3. Problem Statement

Many individuals travel alone in their personal vehicles, resulting in:

- 1.Wasted fuel
- 2.Higher travel costs
- 3.Increased carbon pollution
- 4.Traffic congestion

Difficulty for students and employees to find affordable travel options. There is a need for a simple platform where people can share rides safely and efficiently.

4. Functional Requirements

1. Add Ride

Driver can create a ride by providing name, source, destination, price, and available seats.

2. Search Ride

Users can search for available rides based on destination or route.

3. View All Rides

Shows a list of all posted rides.

4. Book a Seat

Reduces seat count by one when someone books.

5. Delete a Ride

Driver can remove a ride if needed.

6. Simple Console/Web Interface

User-friendly input and display.

5. Non-Functional Requirements

1. **Usability** – Must be easy for beginners to use.
2. **Performance** – Fast search and display of rides.
3. **Reliability** – Data should be stored and processed accurately.
4. **Scalability** – Should allow more features in the future.
5. **Maintainability** – Code must be simple and easy to update.
6. **Security** – Basic input validations and safe data handling.

6. System Architecture

A simple layered architecture:

User Layer – User interacts through console/web interface.

Application Layer – Business logic (add ride, search ride, book ride).

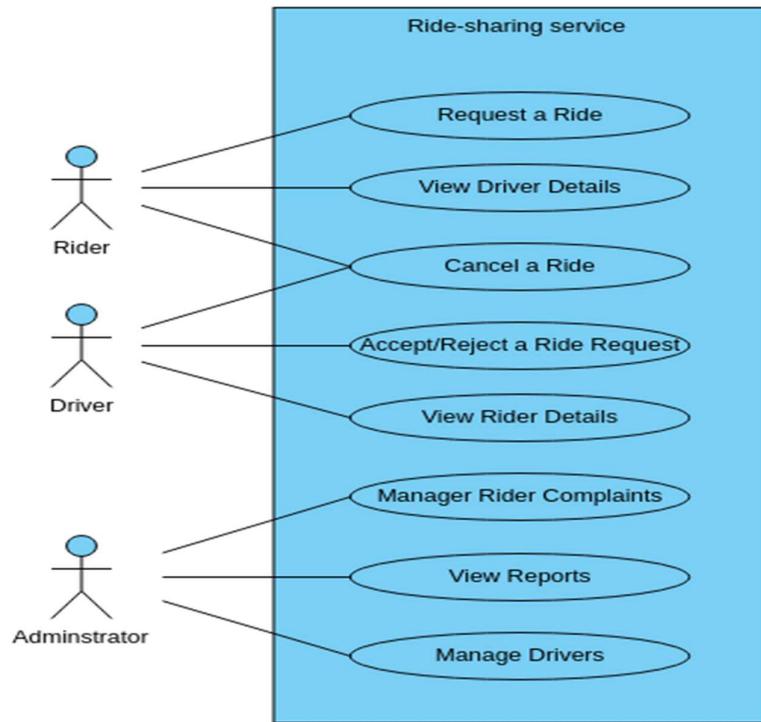
Data Layer – Rides stored in list/dictionary/database (based on implementation).

7. Design Diagrams

a. Use Case Diagram

Actors: User, Driver

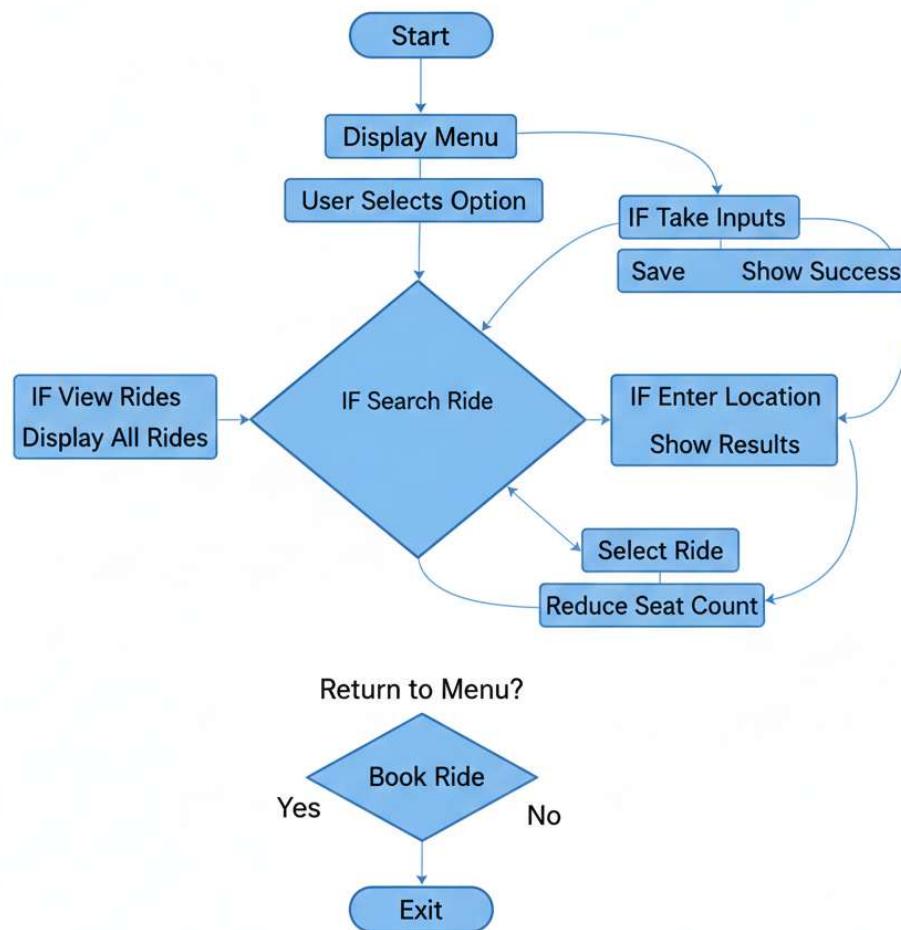
Use Cases: Add Ride, Search Ride, View Rides, Book Ride



b. Workflow Diagram

1. Start →
2. Display Menu →
3. User selects action →
4. Perform action (Add/Search/View/Book) →
5. Show result →
6. Loop to menu →
7. Exit

Carpooling System - Workflow Diagram



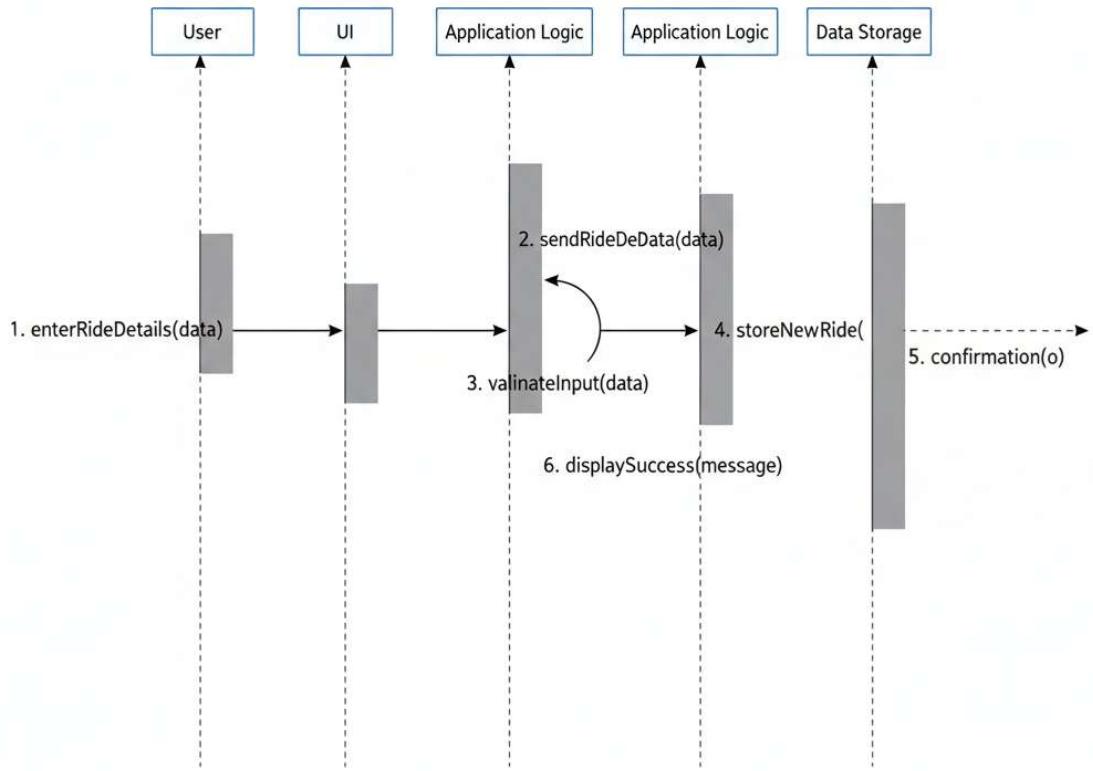
c. Sequence Diagram

User → Application → Data Storage

Example (Add Ride):

User inputs details → System validates → System stores ride → Confirmation sent.

Add Ride System - Successful Flow

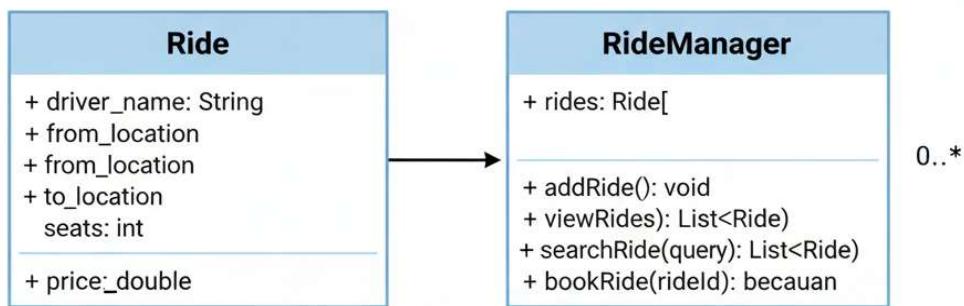


d. Class Diagram

Classes: Ride, RideManager

Components: UI Component, Logic Component, Storage Component

Ride Management System - Class Diagram



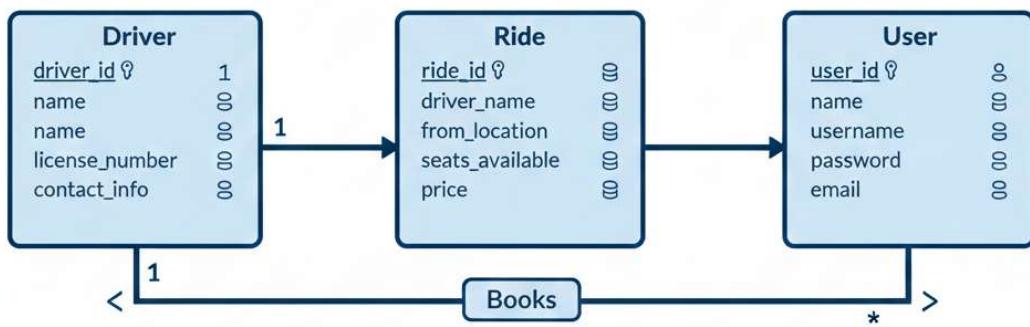
e. ER Diagram

Entities:

Ride (ride_id, driver_name, from, to, seats, price)

Relationships: 1 Ride → Many Users (optional)

Carpooling System - ER Diagram



8. Design Decisions & Rationale

Python chosen for simplicity and readability.

List/Dictionary used for storage to keep the program lightweight.

Modular functions (add_ride, view_rides, search_ride) used for clarity.

Simple text-based UI chosen to make the project easy to run on any system.

9. Implementation Details

The program is implemented in Python using basic data structures. Rides are stored in a list, where each ride is represented as a dictionary. Functions handle adding, searching, viewing, and booking rides. Input validation ensures proper data entry. Program loops until the user exits.

10. Screenshots / Results

```
#This is a "car pooling" programme.  
# in this we can travel with others, by this we can reduce travel expences and pollution too  
# who are travelling solo or having vacant seats in the vehical can use this programme to add others.  
  
rides = []  
#to add a ride  
def add_ride():  
    print("Add a Ride")  
    driver = input("Driver name: ")  
    from_place = input("From: ")  
    to_place = input("To: ")  
    seats = int(input("Seats available: "))  
    price = int(input("Ride price: "))  
    date = input("Date (dd/mm/yyyy): ")  
    time_departure = input("Time of departure (hh:mm): ")  
    ride = {  
        "driver": driver,  
        "from": from_place,  
        "to": to_place,  
        "seats": seats,  
        "price": price,  
        "date": date,  
        "time": time_departure  
    }  
  
    rides.append(ride)  
    print("Ride added successfully")  
#to show available rides  
def show_rides():  
    print("All Rides")  
    if len(rides) == 0:  
        print("No rides available.")  
    else:  
        for counter, r in enumerate(rides, start=1):  
            print(counter, r["driver"], r["from"], "to", r["to"], "Seats:", r["seats"], r["date"], r["time"])  
#to book seat  
def book_seat():  
    show_rides()  
    if len(rides) == 0:  
        return  
    number = int(input("Enter ride number to book: "))  
    if number < 1 or number > len(rides):  
        print("Invalid ride number.")  
  
All Rides  
1 karthik m bhopal to tirupati Seats: 4 21/12/2025 1145  
Enter ride number to book: 1  
Seat booked.  
1.Add Ride  
2.Show Rides  
3.Book Ride  
4.Save Rides  
5.Exit  
Enter choice: 1  
  
All Rides  
1 karthik m bhopal to tirupati Seats: 4 21/12/2025 1145  
Enter ride number to book: 1  
Seat booked.  
1.Add Ride  
2.Show Rides  
3.Book Ride  
4.Save Rides  
5.Exit  
Enter choice: 1
```

Screenshot 1: Book Seat Function

```

    def book_seat():
        if number < 1 or number > len(rides):
            print("Invalid ride number.")
            return

        ride = rides[number - 1]

        if ride["seats"] > 0:
            ride["seats"] -= 1
            print("Seat booked.")
        else:
            print("No seats left in this ride.")

    #to save rides
    def save_rides():
        file = open("rides.txt", "w")
        for r in rides:
            line = r["driver"] + "," + r["from"] + "," + r["to"] + "," + message(r["seats"]) + "," + message(r["price"]) + "," + message(r["date"])
            file.write(line)
        file.close()
        print("Rides saved.")

    #to show available rides
    def load_rides():
        try:
            file = open("rides.txt", "r")
        except:
            print("File not found")

```

The code implements a `book_seat` function that takes a seat number as input. It checks if the seat number is valid (between 1 and the number of rides). If valid, it decrements the seat count for that ride and prints a confirmation message. If the seat count reaches zero, it prints a message indicating no seats left. The code also includes `save_rides` and `load_rides` functions for saving and loading ride data from a file.

Screenshot 2: Main Menu Implementation

```

    def load_rides():
        try:
            file = open("rides.txt", "r")
        except:
            pass
    main_menu()
    def main():
        load_rides()
        while True:
            print("1_Add Ride")
            print("2_Show Rides")
            print("3_Book Ride")
            print("4_Save Rides")
            print("5_Exit")
            choice = input("Enter choice: ")
            if choice == "1":
                add_ride()
            elif choice == "2":
                show_rides()
            elif choice == "3":
                book_seat()
            elif choice == "4":
                save_rides()
            elif choice == "5":
                break
            else:
                print("Invalid choice")

```

The code defines a `main` function that loads ride data from a file. It then enters a loop where it prints a menu with options 1 through 5. Depending on the user's choice, it calls `add_ride`, `show_rides`, `book_seat`, `save_rides`, or exits the loop.

```
gitdemo python project.txt > book_seat
  78     def main():
  79         print("5 Exit")
  80         choice = input("Enter choice: ")
  81         if choice == "1":
  82             add_ride()
  83         elif choice == "2":
  84             show_rides()
  85         elif choice == "3":
  86             book_rides()
  87         elif choice == "4":
  88             save_rides()
  89         elif choice == "5":
  90             break
  91         else:
  92             print("Invalid choice")
  93
  94     main()

All Rides
1 karthik m bhopal to tirupati Seats: 4 2112025 1145
Enter ride number to book: 1
Seat booked.
1. Add Ride
2. Show Rides
3. Book Ride
4. Save Rides
5. Exit
Enter choice: 1
```

11. Testing Approach

1. Unit Testing

Testing individual functions like add_ride(), search_ride().

2. Input Testing

Wrong inputs, blank inputs, invalid numbers.

3. Functional Testing

Checking the correctness of results after each action.

4. User Testing

Tested by classmates to ensure ease-of-use.

12. Challenges Faced

- Designing the structure of ride storage.
- Handling input errors.
- Ensuring the search feature works correctly.
- Creating meaningful diagrams and layout.

13. Learnings & Key Takeaways

- 1.Learned how carpooling helps reduce pollution.
- 2.Improved understanding of Python functions and data structures.
- 3.Learned project documentation and diagram creation.
- 4.Understood how real-world apps manage data and user interaction.
- 5.Enhanced problem-solving and debugging skills.

14. Future Enhancements

- >Add login and authentication.
- >Add online database for real-time updates.
- >Live location tracking.
- >Rating system for drivers and riders.
- >Mobile app version.
- >Payment integration.
- >Pollution reduction meter.

15. References

Python Documentation

Vityarthi essentials in python course

EVS Sustainability course modules

Carpooling concept research papers

Class materials and faculty guidance