**Ride-Sharing App with Driver and Rider Matching**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

Santosh (AP23110011421)

Praveen (AP23110011455)

Venkat (AP23110011456)

Prem Kumar (AP23110011463)



Under the Guidance of

**(Supervisor Name)**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[November 2024]**

# Certificate

Date: 16-Nov-22

This is to certify that the work present in this Project entitled “**Ride-Sharing App with Driver and Rider Matching**” has been carried out by **group** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

**Supervisor**

(Signature)

Prof. / Dr. Kavitha rani

Designation, Affiliation.

**Co-supervisor**

(Signature)

Prof. / Dr. [Name]

Designation, Affiliation.

**Acknowledgements**

We would like to express our gratitude Dr. Kavitha rani for assisting us with the research at every step. Without the advice and assistance of our adviser, this study effort would not be feasible. The suggestions, feedbacks, and encouragement from our advisors were extremely helpful in assisting us in finishing this research.

**Abstract**

This project focuses on the design and implementation of a comprehensive ride-sharing application that enables efficient matching of drivers and riders, revolutionizing urban mobility. The application is built to address key challenges in ride-sharing, including real-time optimization of route and resource allocation, enhancing user experience, and ensuring safety.

The core functionality of the app includes a robust matching algorithm that pairs drivers and riders based on factors such as location, destination, travel time, and user preferences. Advanced geolocation and mapping technologies are integrated to provide precise navigation and seamless ride coordination. The platform also supports dynamic pricing, ensuring fair fare adjustments based on traffic, demand, and trip distance.

The app prioritizes user security through features like two-step identity verification, real-time ride tracking, SOS emergency alerts, and driver/rider rating systems. A sleek and intuitive user interface makes booking rides simple and convenient, while payment gateways ensure secure and flexible transaction options.

Additionally, the application incorporates eco-friendly initiatives by promoting carpooling, thereby reducing traffic congestion and carbon emissions. Through its innovative design, the ride-sharing app seeks to offer a reliable, cost-effective, and sustainable transportation solution while fostering a community-driven approach to mobility.

This system has potential applications in urban centers, corporate transport solutions, and community-based travel networks, making it a versatile tool for modern transportation needs.

**1.Introduction:**

A ride-sharing application with driver and rider matching serves as an innovative solution to modern transportation challenges, offering a seamless platform that connects drivers and riders efficiently. The app leverages advanced matching algorithms and geolocation technology to pair users based on proximity, destination, and route compatibility. By promoting carpooling and optimizing routes, the system aims to reduce travel costs, traffic congestion, and environmental impact. Key features include secure payment integration, dynamic pricing, real-time ride tracking, and safety measures like identity verification and emergency alerts. With a user-friendly interface and robust functionality, this app provides a convenient, cost-effective, and eco-friendly alternative for urban mobility while fostering a sustainable and community-driven approach to transportation.

**Core Classes and Their Key Features:**

1. **Vehicle**
   * **Purpose:** Acts as a base class for all vehicle types.
   * **Key Features:**
     + Contains costPerKm and distance as protected members to define travel costs and distance.
     + Pure virtual function calculateCost() ensures each derived class implements its own cost calculation.
2. **Car, Bike, Taxi (Derived Classes from Vehicle)**
   * **Purpose:** Represent specific vehicle types with unique cost per kilometer.
   * **Key Features:**
     + **Car:** Cost per kilometer is 12.0.
     + **Bike:** Cost per kilometer is 8.0.
     + **Taxi:** Cost per kilometer is 10.0.
     + Each overrides calculateCost() to compute and display the total ride cost.
3. **Driver**
   * **Purpose:** Represents a driver in the ride-sharing app.
   * **Key Features:**
     + Stores the driver's name and vehicle type.
     + Provides a printDetails() function to display driver information.
     + Enables vehicle type matching with riders.
4. **Rider**
   * **Purpose:** Represents a rider in the ride-sharing app.
   * **Key Features:**
     + Stores the rider's name, preferred vehicle type, and destination.
     + Provides a printDetails() function to display rider information.
     + Determines the travel distance based on the destination using getDistanceForDestination().
5. **RideSharingApp**
   * **Purpose:** Manages drivers, riders, and their interactions.
   * **Key Features:**
     + Stores registered drivers and riders in separate lists.
     + Automatically matches riders with available drivers based on vehicle type.
     + Calculates and displays ride costs based on the matched vehicle type and destination distance.
     + Provides functions to add new drivers (addDriver) and riders (addRider), show all registered users, and perform ride matching.

**Display of Registered Users**

1. **Drivers Registered:**
   * The app maintains a list of drivers. Each driver's name and vehicle type are displayed using the showAllDrivers() function. This ensures transparency and clarity about the available options for riders.
2. **Riders Registered:**
   * The app maintains a list of riders, displaying their name, preferred vehicle type, and destination using the showAllRiders() function. This helps in tracking and managing ride requests.

**2.Methodology:**

**Overview of System Structure**

**Core Classes and Concepts**

* **Vehicle**:
  + Abstract base class for all vehicles with attributes for cost per kilometer and travel distance.
  + Defines a pure virtual method calculateCost() to be implemented by derived classes.
* **Car, Bike, Taxi**:
  + Derived classes from Vehicle, each representing a specific type of vehicle.
  + Override the calculateCost() method to compute and display the ride cost based on their unique rates.
* **Driver**:
  + Represents a driver, storing their name and vehicle type.
  + Key for rider-driver matching based on the vehicle type.
* **Rider**:
  + Represents a rider, storing their name, preferred vehicle type, and destination.
  + Includes functionality to calculate the travel distance based on the destination.
* **RideSharingApp**:
  + Central class managing the interaction between drivers and riders.
  + Maintains lists of registered drivers and riders.
  + Performs rider-driver matching and computes ride costs dynamically.

**Junction (Template Class)**

* **Purpose**:
  + A generic class to create connections between entities (e.g., Rider and Driver).
  + Facilitates flexible and reusable matching mechanisms across different system components.
* **Key Functions**:
  + **addEntity**: Add a new entity (Driver/Rider) to the junction.
  + **matchEntities**: Match entities based on specified criteria (e.g., vehicle type).
  + **removeEntity**: Remove an entity from the system.
  + **listEntities**: Display all connected entities.

**Security and Data Integrity**

* **Input Validation**:
  + Ensures valid data is entered for names, vehicle types, and destinations.
  + Prevents invalid or malicious input.
* **Encapsulation**:
  + Driver and Rider information is encapsulated, ensuring controlled access through public methods.
* **Secure Matching**:
  + Matches are conducted only when criteria align, preventing unauthorized or mismatched assignments.
* **Data Logging**:
  + Tracks all interactions, enabling auditing and ensuring accountability.
* **Error Handling**:
  + Robust mechanisms to handle edge cases like missing data or unmatched entities gracefully.

**3.Discussion**

**Input of code**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Vehicle {

protected:

double costPerKm;

double distance;

public:

Vehicle() : costPerKm(0), distance(0) {}

virtual void calculateCost() = 0;

void setDistance(double dist) {

distance = dist;

}

};

class Car : public Vehicle {

public:

Car(double distance) {

costPerKm = 12.0;

this->distance = distance;

}

void calculateCost() override {

cout << "Car ride cost: " << distance \* costPerKm << endl;

}

};

class Bike : public Vehicle {

public:

Bike(double distance) {

costPerKm = 8.0;

this->distance = distance;

}

void calculateCost() override {

cout << "Bike ride cost: " << distance \* costPerKm << endl;

}

};

class Taxi : public Vehicle {

public:

Taxi(double distance) {

costPerKm = 10.0;

this->distance = distance;

}

void calculateCost() override {

cout << "Taxi ride cost: " << distance \* costPerKm << endl;

}

};

class Driver {

private:

string name;

string vehicleType;

public:

Driver(string name, string vehicleType) : name(name), vehicleType(vehicleType) {}

string getVehicleType() const {

return vehicleType;

}

void printDetails() const {

cout << "Driver Name: " << name << endl;

cout << "Vehicle Type: " << vehicleType << endl;

}

};

class Rider {

public:

string name;

string preferredType;

string destination;

Rider(string name, string preferredType, string destination)

: name(name), preferredType(preferredType), destination(destination) {}

void printDetails() const {

cout << "Rider Name: " << name << endl;

cout << "Preferred Vehicle Type: " << preferredType << endl;

cout << "Destination: " << destination << endl;

}

double getDistanceForDestination() const {

if (destination == "Guntur") {

return 50;

} else if (destination == "Vijayawada") {

return 80;

} else if (destination == "Mangalgiri") {

return 150;

} else if (destination == "Tenali") {

return 200;

} else {

return 20;

}

}

};

class RideSharingApp {

private:

vector<Driver> drivers;

vector<Rider> riders;

public:

void addDriver(const Driver& driver) {

drivers.push\_back(driver);

matchAndAssignRide();

}

void addRider(const Rider& rider) {

riders.push\_back(rider);

matchAndAssignRide();

}

void matchAndAssignRide() {

for (auto& rider : riders) {

bool matched = false;

double distance = rider.getDistanceForDestination();

for (auto& driver : drivers) {

if (driver.getVehicleType() == rider.preferredType) {

cout << "\nMatching Driver and Rider found!" << endl;

driver.printDetails();

rider.printDetails();

if (rider.preferredType == "car") {

Car carRide(distance);

carRide.calculateCost();

}

else if (rider.preferredType == "bike") {

Bike bikeRide(distance);

bikeRide.calculateCost();

}

else if (rider.preferredType == "taxi") {

Taxi taxiRide(distance);

taxiRide.calculateCost();

}

matched = true;

break;

}

}

if (!matched) {

cout << "No available driver for rider " << rider.name << " with preferred vehicle " << rider.preferredType << endl;

}

}

}

void showAllDrivers() {

cout << "\nAll Drivers Registered:\n";

for (const auto& driver : drivers) {

driver.printDetails();

}

}

void showAllRiders() {

cout << "\nAll Riders Registered:\n";

for (const auto& rider : riders) {

rider.printDetails();

}

}

};

int main() {

RideSharingApp app;

int choice;

do {

cout << "\n--- Ride Sharing App ---\n";

cout << "1. Add Driver\n";

cout << "2. Add Rider\n";

cout << "3. Show All Drivers\n";

cout << "4. Show All Riders\n";

cout << "5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 1) {

string name, vehicleType;

cout << "Enter driver name: ";

cin >> name;

cout << "Enter vehicle type (car/bike/taxi): ";

cin >> vehicleType;

app.addDriver(Driver(name, vehicleType));

}

else if (choice == 2) {

string name, preferredType, destination;

cout << "Enter rider name: ";

cin >> name;

cout << "Enter preferred vehicle type (car/bike/taxi): ";

cin >> preferredType;

cout << "Enter destination: ";

cin >> destination;

app.addRider(Rider(name, preferredType, destination));

}

else if (choice == 3) {

app.showAllDrivers();

}

else if (choice == 4) {

app.showAllRiders();

}

else if (choice == 5) {

cout << "Exiting the app...\n";

}

else {

cout << "Invalid choice, please try again.\n";

}

} while (choice != 5);

return 0;

}

**Output of code**

--- Ride Sharing App ---

1. Add Driver

2. Add Rider

3. Show All Drivers

4. Show All Riders

5. Exit

Enter your choice: 1

Enter driver name: prem

Enter vehicle type (car/bike/taxi): car

--- Ride Sharing App ---

1. Add Driver

2. Add Rider

3. Show All Drivers

4. Show All Riders

5. Exit

Enter your choice: 2

Enter rider name: venkat

Enter preferred vehicle type (car/bike/taxi): car

Enter destination: Guntur

Matching Driver and Rider found!

Driver Name: prem

Vehicle Type: car

Rider Name: venkat

Preferred Vehicle Type: car

Destination: Guntur

Car ride cost: 600

--- Ride Sharing App ---

1. Add Driver

2. Add Rider

3. Show All Drivers

4. Show All Riders

5. Exit

Enter your choice: 2

Enter rider name: santosh

Enter preferred vehicle type (car/bike/taxi): car

Enter destination: Vijayawada

Matching Driver and Rider found!

Driver Name: prem

Vehicle Type: car

Rider Name: venkat

Preferred Vehicle Type: car

Destination: Guntur

Car ride cost: 600

Matching Driver and Rider found!

Driver Name: prem

Vehicle Type: car

Rider Name: ntosh

Preferred Vehicle Type: car

Destination: Vijayawada

Car ride cost: 960

--- Ride Sharing App ---

1. Add Driver

2. Add Rider

3. Show All Drivers

4. Show All Riders

5. Exit

Enter your choice: 1

Enter driver name: praveen

Enter vehicle type (car/bike/taxi): bike

Matching Driver and Rider found!

Driver Name: prem

Vehicle Type: car

Rider Name: venkat

Preferred Vehicle Type: car

Destination: Guntur

Car ride cost: 600

Matching Driver and Rider found!

Driver Name: prem

Vehicle Type: car

Rider Name: ntosh

Preferred Vehicle Type: car

Destination: Vijayawada

Car ride cost: 960

--- Ride Sharing App ---

1. Add Driver

2. Add Rider

3. Show All Drivers

4. Show All Riders

5. Exit

Enter your choice:

**4.Concluding Remarks:**

The ride-sharing application is a well-structured system designed to streamline the process of connecting riders with drivers efficiently. Through the implementation of core classes such as Vehicle, Driver, and Rider, the system encapsulates essential functionalities while maintaining flexibility and scalability. The inclusion of a template class like Junction enables generic matching and enhances reusability across different components of the application.

Additionally, the methodology prioritizes security and data integrity, ensuring user information is safeguarded, and operations are carried out reliably. With features like dynamic cost calculation, transparent user registration, and automated matching, the system provides a user-friendly and secure platform for ride-sharing.

This application lays a solid foundation for further advancements, such as integrating real-time tracking, payment gateways, or AI-driven route optimization. Overall, it is a robust and adaptable solution that effectively addresses the needs of both riders and drivers in a ride-sharing ecosystem.

**5.Future Work**

The ride-sharing application has significant potential for growth and enhancement. Below are some areas for future work to improve functionality, user experience, and system efficiency:

**1. Integration of Real-Time Features**

Real-Time Tracking:

Implement GPS-based tracking to allow riders and drivers to monitor ride locations in real-time.

Live Availability:

Enable real-time updates on driver availability and estimated arrival times.

**2. Payment System Enhancement**

Online Payment Integration:

Introduce digital payment methods such as credit cards, UPI, and e-wallets.

Dynamic Pricing:

Implement surge pricing based on demand, time of day, or weather conditions.

**3. Advanced Matching Algorithms**

AI and Machine Learning:

Use machine learning algorithms to optimize rider-driver matching based on historical data, preferences, and traffic conditions.

Multi-Rider Matching:

Add support for carpooling to allow multiple riders to share a single ride, reducing costs and environmental impact.

**4. Enhanced User Profiles**

Driver Ratings and Feedback:

Include a rating system for riders and drivers to improve service quality.

Personalized Recommendations:

Offer tailored ride options based on user preferences and ride history.

**5. Scalability and Performance**

Cloud Integration:

Migrate the system to a cloud-based platform for better scalability and availability.

Database Optimization:

Improve data storage and retrieval mechanisms for faster performance, especially as the user base grows.

**6. Security Improvements**

Two-Factor Authentication (2FA):

Add 2FA to enhance account security for riders and drivers.

Data Encryption:

Use end-to-end encryption for sensitive user information and transactions.

**7. Environmental Sustainability**

Electric Vehicle (EV) Support:

Promote eco-friendly transportation by supporting EVs in the fleet.

Carbon Footprint Tracking:

Provide users with insights into the environmental impact of their rides.