**Project Plan for Car-Pooling Management System (CMS)**

**A Digital Solution for Sustainable Urban Mobility**

**BIS698 Information System Project**

**A yellow and purple logo

AI-generated content may be incorrect.**

**Group 12**

**Rahul Reddy Pothireddy**

**Girish Reddy Madireddy**

**Sandeep Reddy Yeddula**

**Charan Kumar Chaganaboina**

**Karunakar Gaddam**

**Problem Statement:**

The system needed to allow both drivers and riders to:

* Post, book, or cancel rides dynamically.
* View updated ride availability in real time.
* Persist booking and ride creation state across sessions.
* Receive immediate, role-specific feedback after key actions (e.g., ride created/booked).
* Track ride and booking history for both drivers and riders.

This required a robust structure that could handle **multi-user interactions**, **role-based UI flows**, and **reliable state maintenance** without loss of data or inconsistency.

**Approach & Solution:**

**1. Ride Data Structure:**

* Each ride stored in a database table with fields:  
  {ride\_id, driver\_id, source, destination, datetime, price\_per\_seat, seats\_available, status}
* Bookings stored separately with {booking\_id, ride\_id, user\_id, status.
* Ensured modularity and normalized schema for easy tracking and updates.

**2. Persistent Booking and Ride State:**

* Bookings and ride posts persisted in a **MySQL database** using foreign key constraints for referential integrity.
* When a rider logs back in, they can:
  + See their previous bookings under "My Bookings"
  + View updated status (Accepted/Rejected/Completed)

**3. Real-Time UI Updates and Feedback:**

* Used **Tkinter (customtkinter)** to provide:
  + Live update of available seats once a booking is made.
  + “Ride created successfully!” message after a driver adds a ride.
  + “Ride booked successfully!” alert after a rider confirms a booking.
* UI also auto-removed rides with 0 seats available, preventing overbooking.

**4. Encapsulation and Role-Based Separation:**

* Created separate classes for Driver, Rider.
  + Each handled their responsibilities (ride creation, booking, request handling).
  + Enabled role-specific dashboards and logic flows.

**5. Ride Validation and Conflict Handling:**

* Before confirming bookings:
  + Checked ride still exists and isn’t full.
  + Updated seats in real time after each booking.
  + Prevented overbooking or double assignment with transactional DB updates.

**6. Enhanced User Experience:**

* Added friendly UI prompts:
  + If no rides: “No rides available. Please check back later.”
  + If no bookings: “You haven’t booked any rides yet.”

**Why It Was Challenging:**

* Built multiple interdependent forms across different user roles (driver, passenger).
* Ensured real-time seat tracking to avoid overbooking.
* Handled **UI feedback** through message boxes to confirm actions.
* Designed **audit tables** to trace actions like ride creation and matching.
* Maintained relational integrity using foreign keys in MySQL (ride\_id, driver\_id).

**Impact:**

* Delivered a **reliable, interactive platform** for drivers and riders.
* Maintained **session-level continuity** and historical visibility for users.
* Improved overall platform **usability, trust, and operational transparency**