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

**A Digital Solution for Sustainable Urban Mobility**

**BIS698 Information System Project**



**Group No: 12**

**Rahul Reddy Pothireddy**

**Girish Reddy Madireddy**

**Sandeep Reddy Yeddula**

**Charan Kumar Chaganaboina**

**Karunakar Gaddam**

**Contents**

**1.**[**Background**](#Background)

**2.** [**Business Problems**](#BUS)

**3.** [**Business Advantages**](#BA)

**4.** [**Proposed System**](#PS)

**5.** [**Modules and Description**](#MD)

**6.** [**System Requirements**](#SR)

**7.** [**Feasibility Analysis**](#FA)

**8.** [**Use case diagram**](#UCA)

**9.** [**Process Model with DFD**](#PMD)

**10.** [**Task list**](#TL)

**11.** [**critical path model**](#CPM)

**12.** [**ERD Diagram**](#ERD)

**13.** [**UI Designs**](#UI)

**14.**[**Conclusion**](#C)

**1.Background**

* **Name**: Car-Pooling Management System (CMS)
* **Location**: Urban Areas (Scalable for Global Use)
* **Organization Type**: Transportation and Mobility Solutions

Urbanization and rising vehicle ownership are major contributors to traffic congestion, higher fuel costs, and environmental degradation. One significant cause is the widespread reliance on private vehicles with single occupants. Carpooling offers a sustainable solution by reducing fuel consumption and carbon emissions. However, the current carpooling process often relies on informal and manual coordination, making it inefficient and unreliable. This project aims to design and implement a digital Car-Pooling Management System (CMS) that automates ride-sharing processes, improves urban mobility, and supports environmental sustainability.

**2.** **Business Problems**

* **Inefficient Coordination**: Traditional carpooling relies on manual coordination through social media, which can lead to confusion and delays.
* **Lack of Automation**: Without automation, matching riders with drivers is inefficient, leading to uncertainty and miscommunication.
* **Limited Communication Channels**: Reliance on informal channels (e.g., WhatsApp) can result in missed messages and scheduling conflicts.
* **Inconsistent Ride Availability**: Traditional systems do not efficiently match riders and drivers, resulting in poor ride availability and last-minute cancellations.
* **Time-Consuming Processes**: Users must invest significant time in re-coordination, which reduces the overall efficiency of carpooling.

**3.** **Business Advantages**

* **Enhanced Ride Matching**: Automated, real-time ride matching ensures better alignment of routes, improving ride availability and reducing delays.
* **Sustainability**: The system supports eco-friendly practices by reducing fuel consumption and carbon emissions through efficient ride-sharing.
* **Cost Efficiency**: Carpooling offers a cost-effective alternative to single-occupancy vehicle usage, reducing fuel expenses for users.
* **Improved User Experience**: A responsive, user-friendly platform ensures ease of use, encouraging more users to adopt carpooling.
* **Scalability**: The CMS can scale to accommodate various urban areas, offering a flexible solution to mobility challenges globally.

**4.** **Proposed System**

The **Car-Pooling Management System (CMS)** will streamline the carpooling process by automating the ride-matching and communication systems, providing users with real-time updates and a structured platform for ride coordination. The CMS will be built using **Python**  for backend services, **MySQL** for secure data management, and a responsive frontend developed using **Tkinter**. The system will allow users to create, join, and manage carpools with ease, enhancing both efficiency and sustainability in urban mobility.

**Key Features of the Proposed System:**

* **Automated Ride Matching**: Efficiently pairs drivers and passengers based on route preferences, improving ride availability and reducing cancellations.
* **Real-Time Updates**: Provides users with live updates about ride status, including cancellations or delays.
* **Secure User Registration and Authentication**: Ensures that only verified users can offer or join rides.
* **Document Verification**: Admins can verify the eligibility of drivers’s by checking documents such as car licenses and insurance.
* **User-Friendly Interface**: The system provides an intuitive and responsive interface for both riders and passengers, ensuring ease of use for all.

**5.** **Modules and Description**

* **Admin Module**: The admin module enables administrators to manage user accounts, ride data, and ensure the integrity of the system by verifying documents and managing user activity.
* **Driver Module**: Drivers can register, offer rides, and upload necessary documents (e.g., car license and insurance) to ensure safety. They can specify ride details, such as departure time, destination, and available seats.

**User Module**: Passengers can register, search for rides along their desired routes, request rides, and track the status of their trips. They can also cancel or modify ride requests and view ride details in real-time

**6.** **System Requirements**

**Hardware Requirements:**

* **Processor**: Intel Core i3 or equivalent (minimum)
* **RAM**: 4 GB (minimum)
* **Storage**: 10 GB free space (minimum)

**7.** **Feasibility Analysis**

**Economic Feasibility:** The breakdown of costs is as follows:

• **Software Development**: $7,000  
• **Database Hosting (MySQL Server)**: $50 per month = $600 annually  
• **Cloud Storage for Data**: $120 per year  
• **System Installation & Training**: $1,000  
• **Ongoing Maintenance & Support**: $80 per hour (estimated 15 hours annually = $1,200)

**Total estimated cost**: $9,920 for the first year.

**8.****Use Case Diagrams**

**Use Case 1: Register User**

| **Use Case Name: Register User** | **ID: UC-1** | **Priority: High** |
| --- | --- | --- |
| **Action:** User | | |
| **Description:** A new user (either a Driver or User) registers on the platform by providing necessary details. | | |
| **Trigger:** User wants to create an account on the CMS platform. | | |
| **Type:** ☑ External ☐ Temporal | | |
| **Preconditions:** | | |
| 1. The user is not already registered. | | |
| 2. The CMS platform is online. | | |
| **Normal Course:** | | |
| 1. User selects "Register" on the homepage. | | |
| 2. User enters personal details (name, email, phone number).  3. User sets a password. | | |
| 4. System verifies email and phone number. | | |
| 5. User account is created and stored in the User Database (D1). | | |
| **Postconditions:** | | |
| 1. User account is created and stored in the User Database (D1). | | |
| 2. User can log in to the system. | | |

**Use Case 2: Create Ride**

| **Use Case Name: Create Ride** | **ID: UC-2** | **Priority: High** |
| --- | --- | --- |
| **Action:** Driver | | |
| **Description:** A Driver creates a ride offer by specifying details such as starting point, destination, departure time, and available seats. | | |
| **Trigger:** Driver wants to offer a ride. | | |
| **Type:** ☑ External ☐ Temporal | | |
| **Preconditions:** | | |
| 1. Driver is logged in. | | |
| 2. Driver has uploaded necessary documents (license, insurance). | | |
| **Normal Course:** | | |
| 1. Driver selects "Create Ride" | | |
| 2. Driver enters ride details (starting point, destination, departure time, available seats). | | |
| 3. System validates the ride details. | | |
| 4. Ride offer is stored in the Rider Database . | | |
| 5. users are notified of the new ride offer. | | |
| **Postconditions:** | | |
| 1. Ride offer is stored in the Rideer Database. | | |
| 2. users can view and request the ride. | | |

**Use Case 3: Search and Request Ride**

| **Use Case Name: Search and Request Ride** | **ID: UC-3** | **Priority: High** |
| --- | --- | --- |
| **Action:** User | | |
| **Description:** A User (passenger) searches for available rides and requests a ride. | | |
| **Trigger:** User wants to find a ride. | | |
| **Type:** ☑ External ☐ Temporal | | |
| **Preconditions:**  1. User is logged in.  2. Ride offers are available in the system. | | |
| **Normal Course:** | | |
| 1. User selects "Search Ride." | | |
| 2. User enters search criteria (starting point, destination, time). | | |
| 3. System displays matching ride offers. | | |
| 4. User selects a ride and requests it. | | |
| 5. System notifies the Driver of the request. | | |
| **Postconditions:** | | |
| 1. Ride request is stored in the Ride Request Database (D3). | | |
| 2. Driver is notified of the request. | | |

**Use Case 4: Admin Manage Users**

| **Use Case Name: Admin Manage Users** | **ID: UC-4** | **Priority: Medium** |
| --- | --- | --- |
| **Action:** Admin | | |
| **Description:** The admin manages user accounts, including activating, blocking, or deleting accounts. | | |
| **Trigger:** Admin needs to manage user accounts. | | |
| **Type:** ☑ External ☐ Temporal | | |
| **Preconditions:** | | |
| 1. Admin is logged in. | | |
| 2. User accounts exist in the system. | | |
| **Normal Course:** | | |
| 1. Admin selects "Manage Users." | | |
| 2. Admin views a list of all users. | | |
| 3. Admin selects a user and chooses to activate, block, or delete the account. | | |
| 4. System updates the user's status in the User Database (D1). | | |
| **Postconditions:** | | |
| 1. User account status is updated in the User Database (D1). | | |
| 2. User receives a notification of the change (if applicable). | | |

**Use Case 5: Driver Upload Documents**

| **Use Case Name: Driver Upload Documents** | **ID: UC-5** | **Priority: Medium** |
| --- | --- | --- |
| **Action:** Driver | | |
| **Description:** A Driver uploads necessary documents (e.g., driver’s license, insurance) to verify their eligibility to offer rides. | | |
| **Trigger:** Driver wants to verify their account. | | |
| **Type:** ☑ External ☐ Temporal | | |
| **Preconditions:** | | |
| 1. Driver is logged in. | | |
| 2. Driver has not yet uploaded documents. | | |
| **Normal Course:** | | |
| 1. Driver selects "Upload Documents." | | |
| 2. Driver uploads required documents. | | |
| 3. System verifies the documents. | | |
| 4. Documents are stored in the Document Database (D4). | | |
| **Postconditions:** | | |
| 1. Documents are stored in the Document Database (D4). | | |
| 2. Driver is marked as verified. | | |

**9.** **Process Model with DFD**

The Data Flow Diagram (DFD) will illustrate the flow of data within the system. Below is a description of the Context Diagram and Level-0 Diagram:

**Context Diagram:**

**A diagram of a car-peeling management system

Description automatically generated**

**Explanation:**

This diagram represents the architecture of a **Car-Pooling Management System** and its interactions with different users. Here’s an explanation of each component and how they interact:

1. **Passenger**
   * Requests a ride from the system.
   * Receives confirmation once a driver is assigned.
2. **Driver**
   * Posts a ride offering to the system.
   * Receives confirmation when a passenger is assigned.
3. **Car-Pooling Management System** (Central System)
   * Acts as the core system managing ride requests and driver postings.
   * Matches passengers with drivers.
   * Confirms rides for both parties.
4. **Admin**
   * Verifies drivers before they can offer rides.
   * Grants permission to verified drivers to operate on the system.

**Workflow:**

* A **passenger** requests a ride in the system.
* A **driver** posts a ride offering.
* The system matches them and sends **ride confirmation** to both.
* The **admin** verifies new drivers and grants them access to post rides.

This system ensures safe and efficient carpooling by verifying drivers and managing ride-sharing operations.

**A diagram of a software system

Description automatically generatedLevel 0 Diagram**

**Explanation:**

**Passenger/User: Interacts with the system to register, login, search/post rides, and place orders.**

**Admin: Manages driver status and system data.**

**Processes**

**1.0 Save User Data: Handles user registration and stores data in D1 User Data.**

**2.0 Authenticate User: Validates credentials during login using D1 User Data.**

**3.0 Manage Ride Data: Processes ride creation/updates/deletion and stores in D2 Ride Data.**

**4.0 Process Orders: Saves booking details to D3 Order Data.**

**5.0 Confirm Order: Fetches order data for display and confirmation.**

**Data Stores**

**D1 User Data: Stores user credentials and profiles.**

**D2 Ride Data: Contains ride details (e.g., locations, timings).**

**D3 Order Data: Stores passenger bookings and payment info.**

**Data Flows**

**User → 1.0: Enter Data (registration).**

**User → 2.0: Enter Credentials (login).**

**User → 3.0: Search Ride / Post Ride, Add/Update/Delete Ride.**

**User → 4.0: Place Order.**

**User - 5.0: Confirm Order (display and confirmation).**

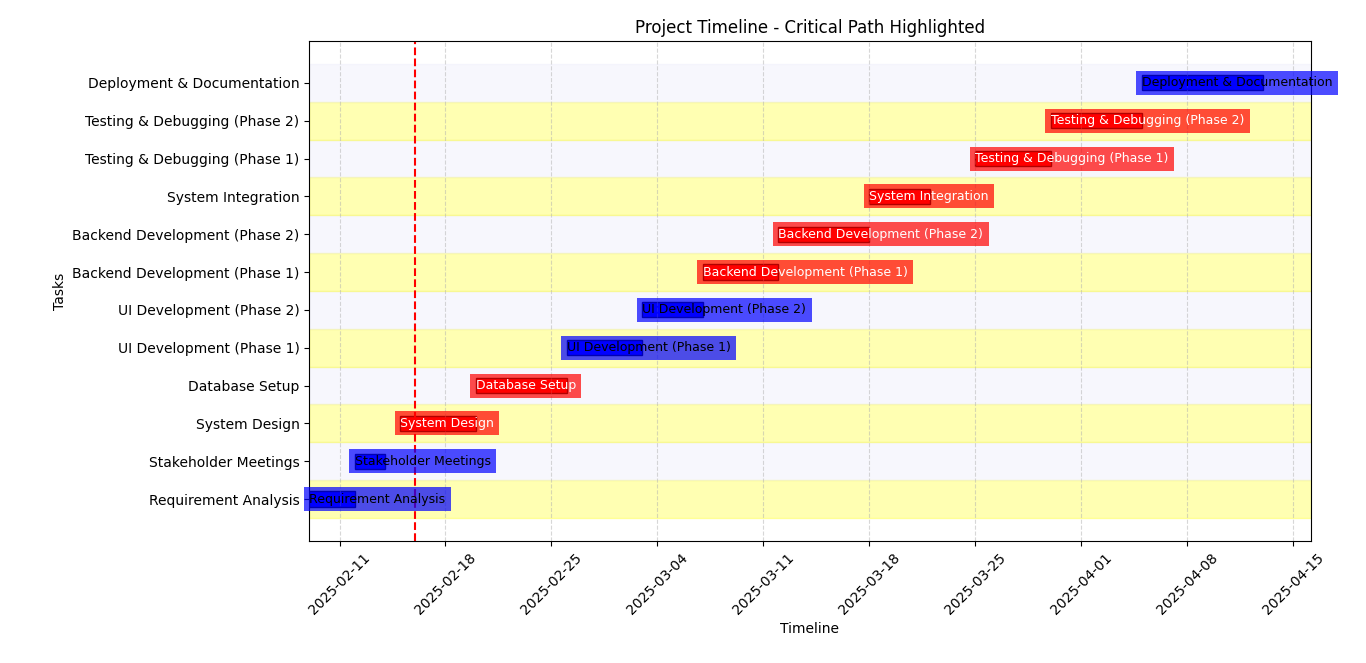
**Admin → 3.0: Enable/Disable Driver (updates D2 Ride Data).**

**10.** **Task List**

| **Week** | **Task Name** | **Description** | **Assigned To** |
| --- | --- | --- | --- |
| Week 1 | Requirement Analysis | Gather business needs, define project scope, and identify key functionalities. | Girish |
| Week 2 | Stakeholder Meetings | Discuss project goals, constraints, and expectations with stakeholders. | Charan |
| Week 3 | System Design | Develop system architecture, DFDs, ERD, and UI wireframes. | girish |
| Week 4 | Database Setup | Implement MySQL database schema and define relationships. | karunakar |
| Week 5 | UI Development (Phase 1) | Design user interfaces for Driver, User, and Admin dashboards. | Rahul |
| Week 6 | UI Development (Phase 2) | Implement navigation, styling, and input forms for all modules. | Charan |
| Week 7 | Backend Development (Phase 1) | Develop core functionalities: user registration, authentication, and ride posting. | Karunakar |
| Week 8 | Backend Development (Phase 2) | Implement ride matching, real-time updates, and trip management. | Charan |
| Week 9 | System Integration | Connect frontend, backend, and database for seamless functionality. | Sandeep |
| Week 10 | Testing & Debugging (Phase 1) | Conduct functional and usability testing for all modules. | Sandeep |
| Week 11 | Testing & Debugging (Phase 2) | Identify and resolve system bugs, optimize performance. | karunakar |
| Week 12 | Deployment & Documentation | Deploy the final system and prepare user manuals and technical documentation. | Rahul |

#### ****11.**** ****Critical Path Model****

The critical path model will show the sequence of tasks and their dependencies. Below is a simplified critical path:



The critical path in project management represents the longest sequence of dependent tasks that determines the minimum project duration. Any delay in these tasks will directly impact on the project’s completion date. In the provided Gantt chart, red-colored tasks indicate critical activities such as System Design, Database Setup, Backend Development, System Integration, and Testing & Debugging. To ensure timely project delivery, these tasks must be closely monitored and completed on schedule.

**12.****ERD Diagram**

A screenshot of a graph

AI-generated content may be incorrect.

This database schema represents a ride-sharing application with three main entities: Users, Rides, and Bookings. The **Users** table stores information about all users, including their username, contact details, and a flag is\_driver to distinguish drivers from passengers. The **Rides** table contains details about the rides offered by drivers, such as the starting location, destination, departure and arrival times, number of available seats, and price per seat. Each ride is linked to a driver through the driver\_id, which is a foreign key referencing the Users table. The **Bookings** table captures reservations made by passengers for specific rides. It includes a reference to both the ride (ride\_id) and the passenger (passenger\_id), linking back to the Rides and Users tables respectively. The status field in Bookings indicates the current state of a booking (e.g., pending or confirmed), and created\_at stores the timestamp of when the booking was made. The relationships between these tables enable a single user to offer multiple rides as a driver and to make multiple bookings as a passenger. This design supports key functionality for a ride-sharing platform, ensuring data consistency and relational integrity.

**13.****UI Designs**

**Home Page**

A screenshot of a car connect

AI-generated content may be incorrect.

**Login**

A screenshot of a login page

AI-generated content may be incorrect.

**Register As User or Driver**

A computer screen shot of a registration form

AI-generated content may be incorrect.

**User Dasboard**

A screenshot of a computer

AI-generated content may be incorrect.

**Driver dashboard**

A screenshot of a computer

AI-generated content may be incorrect.

Driver Report

A screenshot of a report

AI-generated content may be incorrect.

**14.** **Conclusion**:

The Car-Pooling Management System offers a comprehensive solution to the inefficiencies of traditional carpooling methods. By automating ride matching and improving communication between drivers and passengers, CMS promotes sustainable urban mobility and enhances the overall commuting experience. With its scalable design and future enhancements, CMS is poised to become a key player in smart transportation initiatives, contributing to a greener and more efficient urban future.