Malnad College of Engineering

(An autonomous Institution under Visvesvaraya Technological University,Belgaum)

PB#21, Hassan -573202, Karnataka, India

Full Stack Development (23IS505)

TITLE: "Carpooling :A smarter way to travel"

Group number: 06

Submitted by

Name	USN
Harshitha S	**4MC23IS042**
Inchara R Javagal	**4MC23IS046**
Janhavi T R	**4MC23IS047**
Manya B	**4MC23IS058**

Under the guidance of

Mr. Krishna Swaroop A

Assistant Professor,

Department of ISE

Department of Information Science & Engineering

Malnad College of Engineering

PB# 21, Hassan - 573 201, Karnataka, India

Date of submission: 27-11-2025

Table of Content

1.Abstract

The Car Pooling System is a web-based platform developed using Django to simplify and modernize daily commuting by enabling users to share rides efficiently. Traditional travel often relies on personal vehicles or public transportation, leading to increased fuel consumption, traffic congestion, and environmental pollution. Additionally, coordinating shared rides manually is highly unreliable, as it depends on verbal communication or informal messaging, which may result in confusion, safety concerns, and lack of proper organization. This project aims to develop a digital solution where users can register, post rides, search for available trips, book seats, and manage their travel schedules in a structured and convenient manner.

The platform incorporates secure login access for both drivers and passengers, ensuring authenticated use of the system. Users can create detailed ride listings including locations, travel times, vehicle information, and the number of available seats. Passengers can browse these listings and reserve seats instantly. The system records all trips, user interactions, and bookings in a centralized database, allowing efficient data retrieval and management. The web application was designed using Django as the backend framework and developed in Visual Studio Code, utilizing HTML, CSS, and Bootstrap for the frontend interface.

Additionally, the platform enhances safety and trust among users through account verification and transparent ride information. The database-driven structure ensures consistency and prevents data manipulation. The system supports smooth navigation, fast response time, and reliable storage of ride information. The Car Pooling System provides an environmentally friendly, cost-effective, and time-saving solution that encourages shared mobility. Overall, this web application offers a modernized travel experience that supports sustainable transportation practices and reduces the overuse of private vehicles.

2.Introduction

Transportation plays a vital role in the daily lives of individuals, particularly students and working professionals who commute regularly. However, the increasing number of vehicles on the road has resulted in overcrowded traffic conditions, rising fuel prices, and greater environmental impact. Carpooling has emerged as an effective solution for these challenges by

enabling people traveling on similar routes to share a single vehicle. Despite its benefits, most carpooling arrangements are still made informally through personal contacts or social media groups, leading to coordination difficulties, unreliability, and discontinuity.

To address these issues, a digital Car Pooling System was conceptualized and developed using Django. The system provides an organized and dependable platform that connects drivers offering rides with passengers seeking transportation. By automating the ride-booking process, the platform eliminates the complications associated with manual coordination. The system's domain was chosen because carpooling is a globally relevant need and holds substantial potential for reducing traffic congestion and environmental degradation.

In real-world scenarios, people often struggle to coordinate ride sharing due to time mismatches, lack of communication channels, or limited visibility of available rides. Moreover, there is no structured way to maintain travel records or verify user details, resulting in safety concerns. The Car Pooling System addresses these challenges by offering secure authentication, user profiles, detailed ride listings, and an automated booking mechanism. The system enhances ease of use, improves trust, and supports efficient travel planning. It represents a meaningful application of web technologies in solving real-world mobility problems.

This project provides an opportunity to apply backend and frontend development skills, database management techniques, and real-world problem-solving principles. By implementing the system through Django in Visual Studio Code, the project demonstrates practical understanding of MVC architecture, session handling, form validation, and UI design. The Car Pooling System thus modernizes traditional ride sharing by creating a digital ecosystem that is organized, secure, and user-friendly.

3.Objectives of the Project

The primary objective of the Car Pooling System is to create an efficient digital platform that facilitates shared travel by connecting drivers and passengers. The system aims to reduce dependency on individual vehicles, lower travel expenses, and encourage eco-friendly commuting practices. It seeks to automate the entire ride-sharing process, making it easier for users to find, post, and book rides without manual coordination.

A key objective is to ensure secure user authentication so that only registered users can post or book rides. Another goal is to maintain a centralized and organized record of rides, bookings,

travel routes, and user information. The project also focuses on providing an intuitive interface where users can interact with the system, view available rides, and manage their bookings effortlessly. Ultimately, the platform is designed to improve convenience, save time, enhance road efficiency, and promote sustainability through ride sharing.

4. System Requirements

4.1 Software Requirements

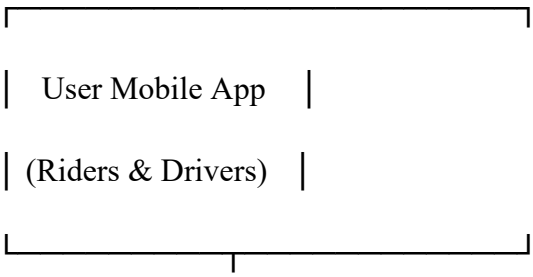
The Car Pooling System was developed using Python as the core programming language, with Django serving as the backend framework due to its built-in ORM, authentication system, and modular architecture. Visual Studio Code was used as the primary IDE for development, offering support for Django extensions, syntax highlighting, and integrated terminal functionalities. SQLite was used as the default database to store ride details, booking records, and user information. The frontend was created using HTML, CSS, Bootstrap, and Django templates, allowing responsive and efficient UI design. Additional tools such as Django forms, static file handling, and Bootstrap components were used to achieve a clean and organized interface.

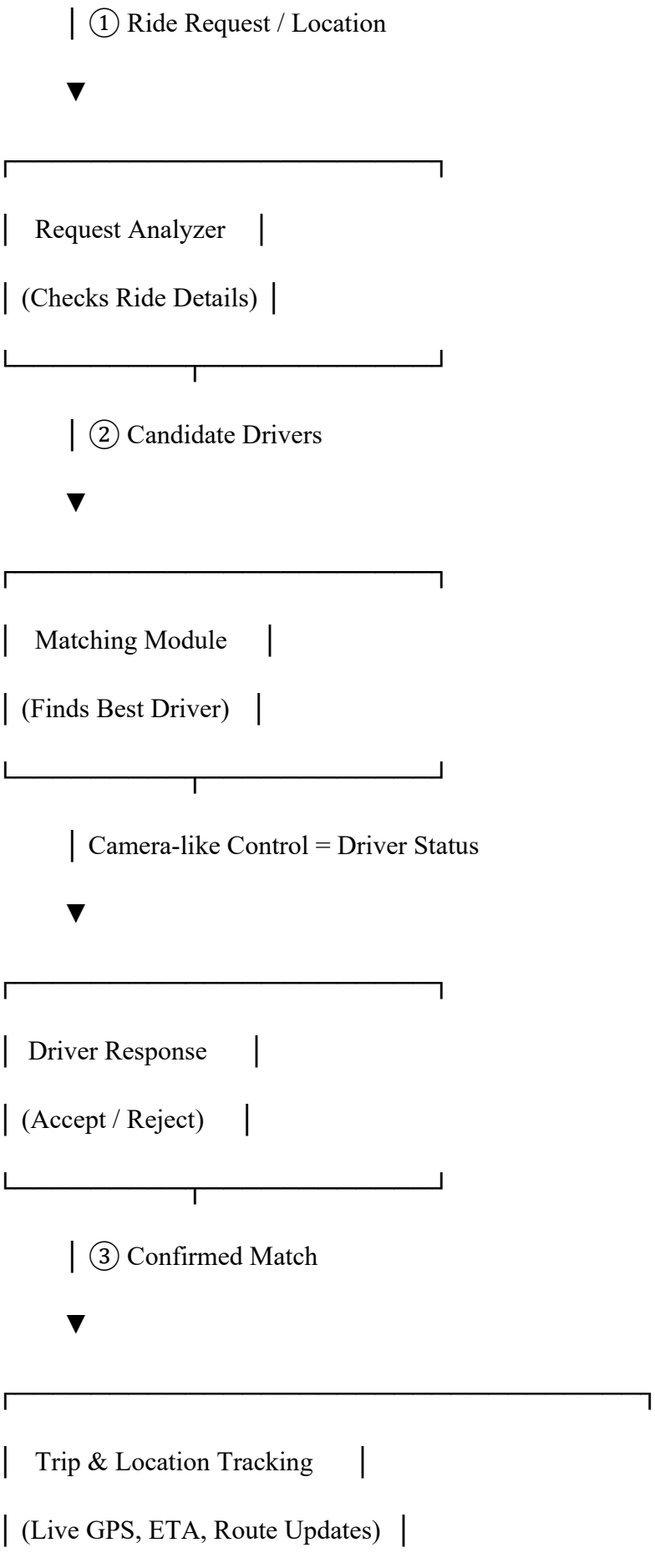
4.2 Hardware Requirements

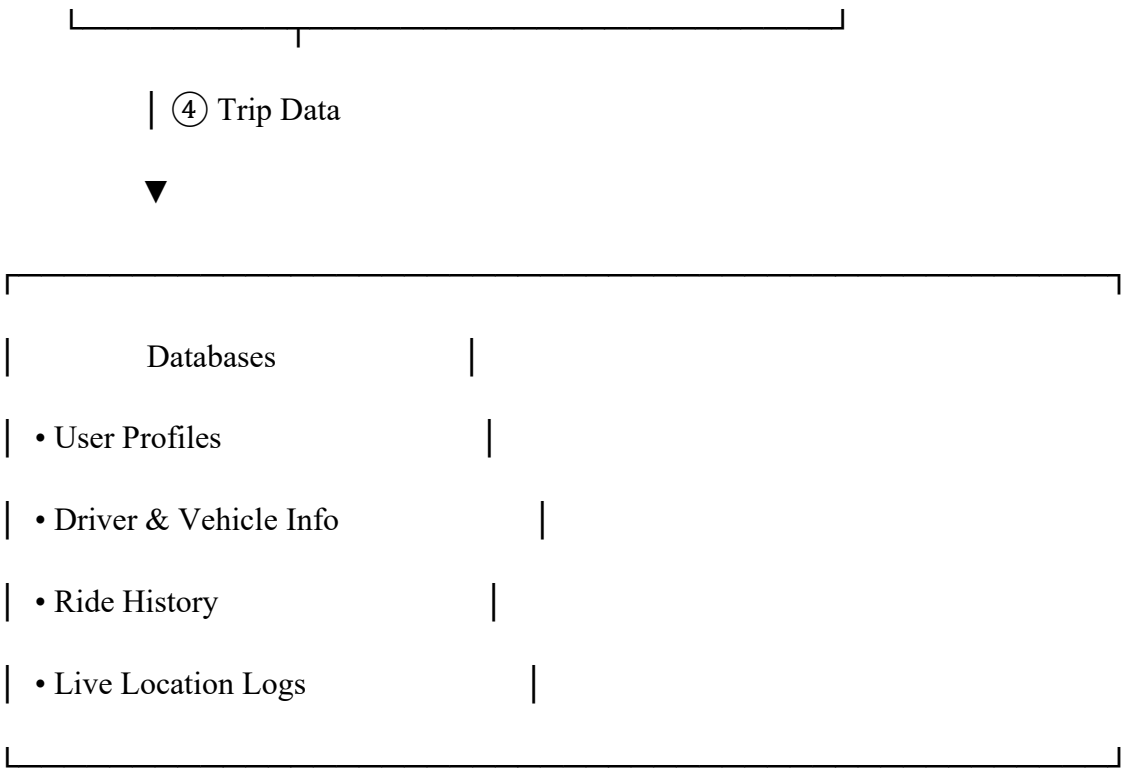
The system requires a computer with a minimum of 4 GB RAM and an Intel i3 or equivalent processor to run the Django development server smoothly. At least 10 GB of free disk space is needed for storing project files, logs, and database entries. A stable internet connection is required only for optional deployment, not for local development. The hardware requirements are minimal, making the system accessible for most environments.

5. System Design

5.1 Architecture Diagram




① Ride Request / Location
Request Analyzer
(Checks Ride Details)
② Candidate Drivers
Matching Module
(Finds Best Driver)
Camera-like Control = Driver Status
Driver Response
(Accept / Reject)
③ Confirmed Match
Trip & Location Tracking
(Live GPS, ETA, Route Updates)



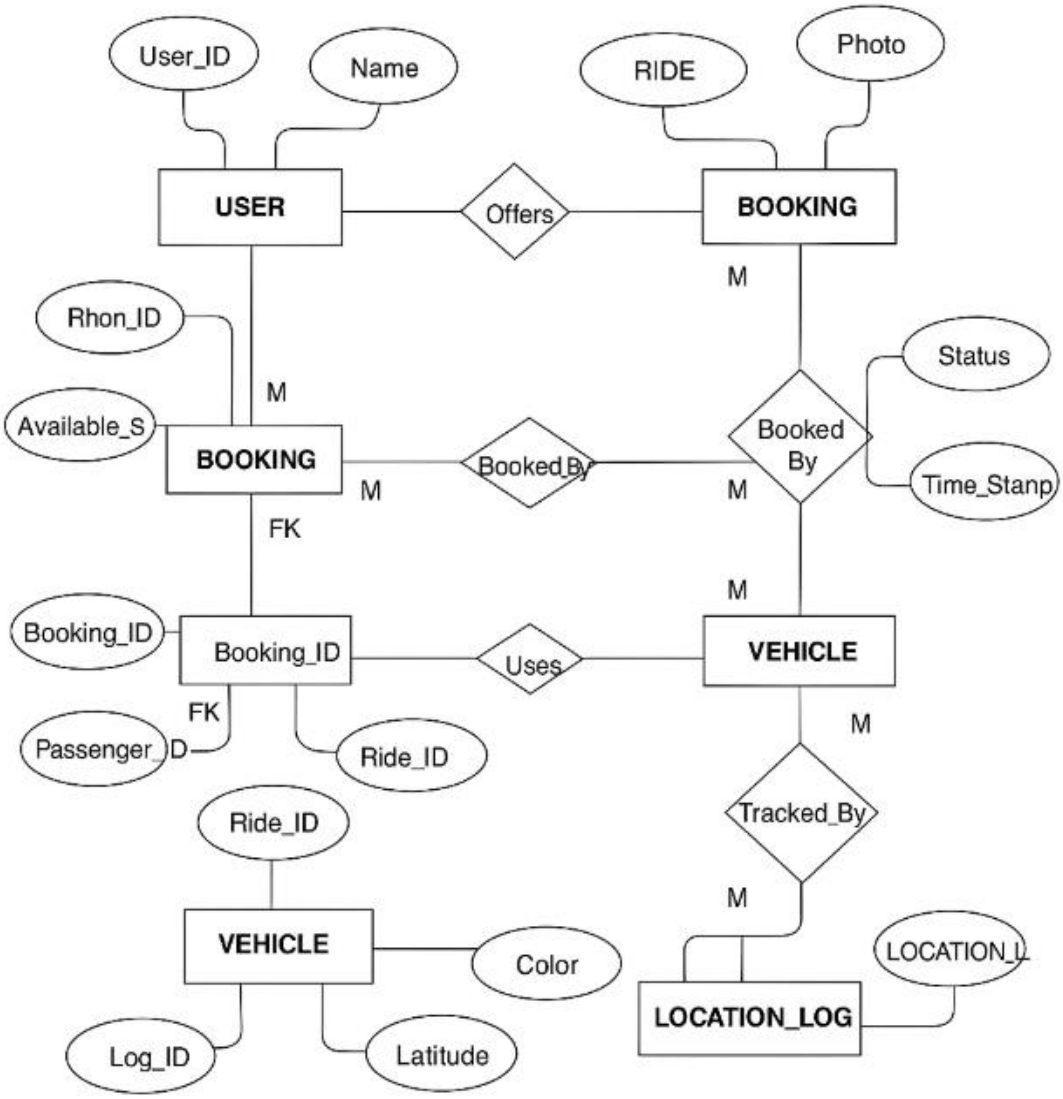
5.2 Explanation of Architecture

The Car Pooling System follows Django's Model-View-Template architecture. The user interacts with the interface through a web browser, where ride listings, forms, and booking options are presented. Requests from the user are directed through Django's URL routing system, which identifies the appropriate view to process each action. The view contains the logic for posting rides, displaying available rides, validating user inputs, and handling bookings.

The view communicates with the model, which represents the database structure of rides, users, and bookings. Once the view retrieves or updates data from the model, it passes the processed information to the template, which renders the final HTML output. The template uses Bootstrap for styling and Django tags for dynamic content rendering. This separation of concerns ensures scalability, readability, and ease of maintenance.

6. Database Design

ER Diagram



ER Diagram Description

The Entity–Relationship (ER) diagram represents the core components of the Carpooling System, illustrating how users, rides, bookings, vehicles, and location logs interact with one another. The diagram models the real-world flow of offering rides, booking seats, and tracking vehicle locations in a systematic and relational manner.

1. USER Entity

The **USER** entity represents individuals registered in the system. Each user can act either as a **Driver** who offers rides or as a **Passenger** who books available rides through the platform.

Attributes

- **User_ID** – Primary Key; unique identifier for each user
- **Name** – Full name of the user

Relationship

- One USER can offer **multiple rides** (1-to-Many with RIDE).
- One USER can make **multiple bookings** (1-to-Many with BOOKING). Thus, a single user can participate in the system in multiple roles.

2. RIDE Entity

The **RIDE** entity represents the transportation service offered by a driver. Each ride contains essential details such as available seats and route information.

Attributes

- **Ride_ID** – Primary Key; unique identifier for the ride
- **Photo / Ride Details** – Additional information about the ride, driver, or vehicle

Relationship

- A USER **offers** multiple rides → **1-to-Many** relationship
- A RIDE can receive multiple BOOKINGS → **1-to-Many** relationship with BOOKING
- A RIDE is associated with exactly one VEHICLE

3. BOOKING Entity

The **BOOKING** entity stores seat reservation details made by passengers. This entity enables the connection between USERS and RIDES, forming the many-to-many relationship of users booking different rides.

Attributes

- **Booking_ID** – Primary Key
- **Phone_ID / Status / Timestamp** – Metadata representing device used, current status of booking, and booking time
- **Available_Seats** – Remaining seats in the ride after booking

Relationships

- **Booked_By** → Connects USER ↔ BOOKING
- **Booked** → Connects RIDE ↔ BOOKING

This creates:

- **M:N (Many-to-Many) relationship** between USER and RIDE → implemented through the BOOKING entity.

Additionally:

- A BOOKING references the VEHICLE assigned to the corresponding ride → **Many-to-One** relationship (many bookings may use the same vehicle).

4. VEHICLE Entity

The **VEHICLE** entity represents the registered vehicle used by the driver for ride-sharing. It stores details required for identification and real-time tracking.

Attributes

- **Vehicle_ID** – Primary Key
- **Color** – Vehicle color
- **Latitude** – Current vehicle location (updated during tracking)

Relationship

- A VEHICLE is associated with multiple RIDES over time

- A BOOKING “uses” a VEHICLE → Each booking references the vehicle used by that ride

This supports proper assignment and tracking of vehicles used across rides.

5. LOCATION_LOG Entity

The **LOCATION_LOG** entity maintains historical and real-time location coordinates for a particular vehicle.

Attributes

- **Log_ID** – Primary Key
- **Latitude / Longitude** – Exact geographical coordinates of the vehicle

Relationship

- A VEHICLE produces multiple LOCATION_LOG entries → **1-to-Many** relationship

This enables continuous tracking and route reconstruction if required

Tables Description

The database includes three primary tables: User, Ride, and Booking. The User table contains details such as username, email, password, and profile information. The Ride table stores ride-related data including source, destination, date, time, available seats, and driver ID. The Booking table maintains information about booked seats, linking each booking to a ride and a user through foreign keys. This design ensures consistency and avoids data duplication while supporting efficient querying.

7. Implementation

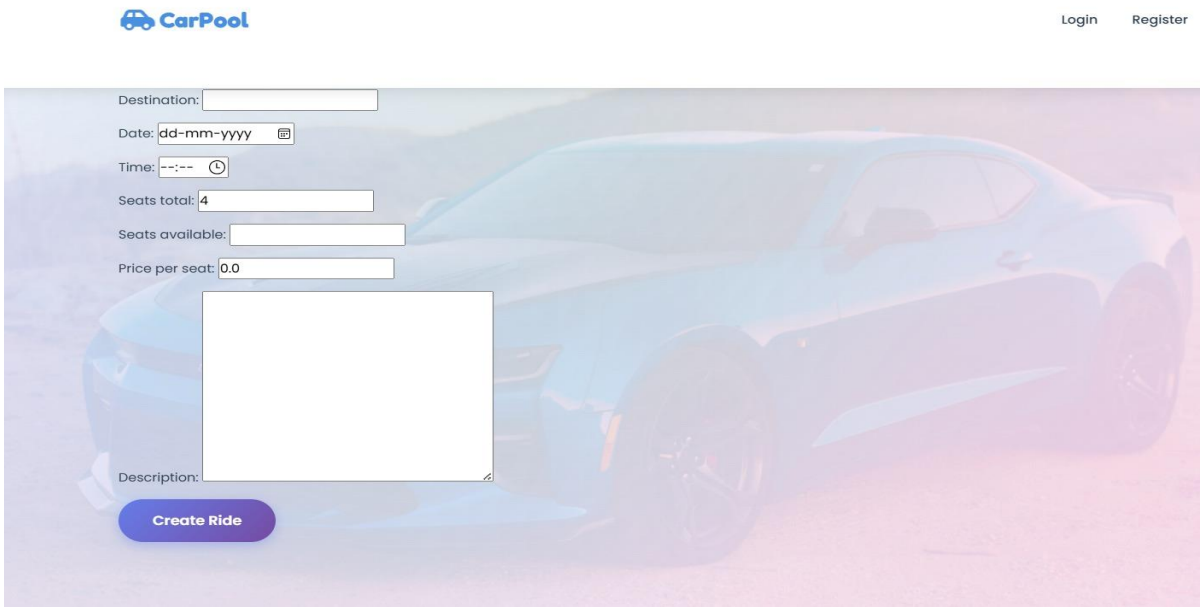
The implementation involved defining Django models for users, rides, and bookings. Views were created to manage posting rides, editing details, browsing listings, and booking seats. URL patterns were established to map frontend requests to specific view functions. Templates were developed using HTML and Bootstrap to create a clean interface for the homepage, ride-list, add-ride, ride-details, and booking pages. The Django ORM handled all database

operations, simplifying data management. The system was thoroughly tested using Django's development server and VS Code's integrated terminal.

8. Webpage Development Using Django in Visual Studio Code

The entire web interface was created using Visual Studio Code, where all template files, static resources, and Python scripts were organized within Django's project structure. The editor allowed easy navigation between models, views, and templates. The ride posting form, booking form, home page, and ride listing pages were built inside the templates folder using HTML and Django template tags. The integrated terminal in VS Code enabled running the Django server, performing migrations, and debugging errors efficiently. Visual Studio Code made the development process smooth, allowing rapid testing and refinement of the webpage.

9. Screenshots



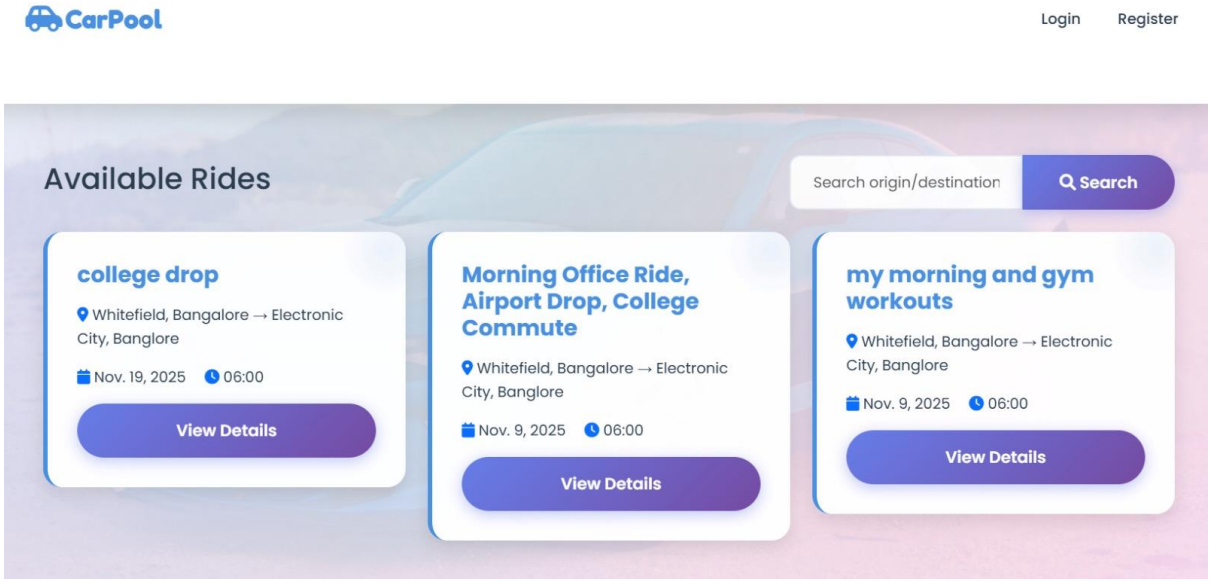
CarPool

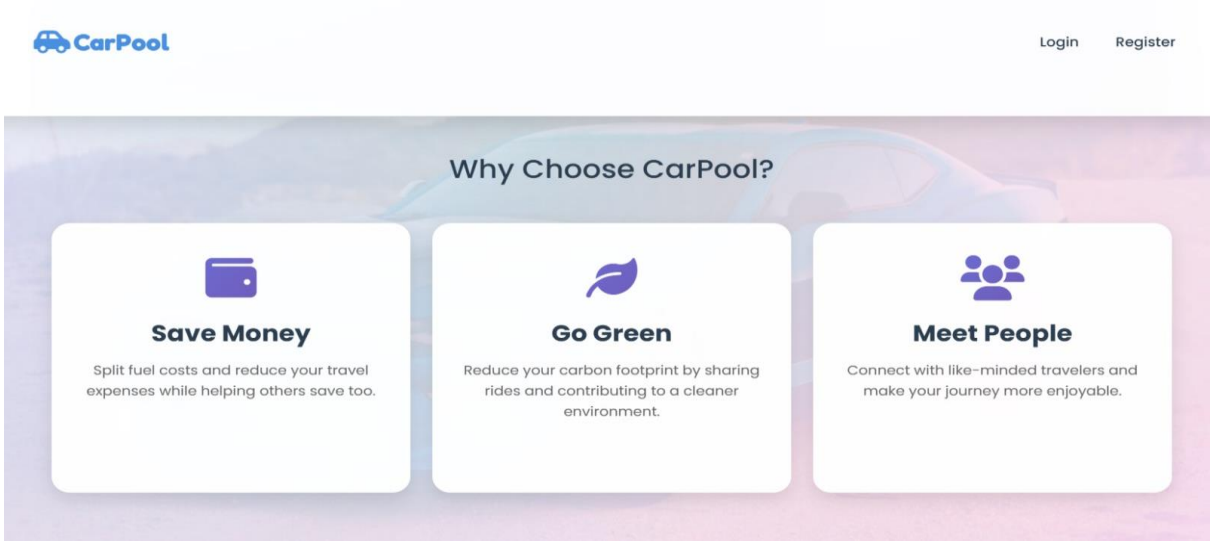
Password:

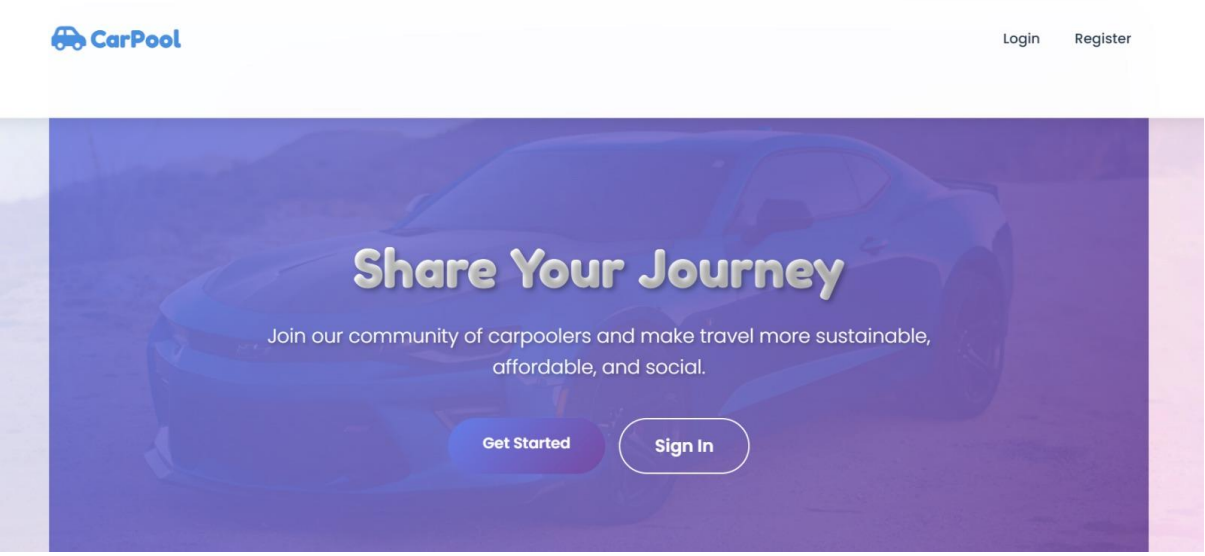
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Register







10. Testing

The Car Pooling System underwent form validation testing, login testing, booking testing, and navigation testing. All modules were verified to ensure correct functionality. The system successfully prevented invalid ride entries, rejected bookings when seats were unavailable, and handled incorrect login attempts appropriately. Thorough testing ensured reliable and stable performance.

11. Results

The system successfully enables users to offer and book rides through a simple, organized, and secure platform. It reduces manual effort and eliminates the communication gaps seen in traditional ride-sharing. The system meets all project objectives by providing efficient ride management, secure authentication, and easy data retrieval. The platform encourages shared mobility and supports sustainable travel.

12. Conclusion

The Car Pooling System demonstrates an effective application of Django for solving real-world transportation challenges. By digitizing ride sharing, the system improves convenience, reduces travel costs, and supports environmental sustainability. The project strengthened understanding of full-stack development, database design, and user interface creation. It stands as a reliable and user-friendly solution that can be expanded into a professional mobility service.

13. Future Enhancements

Future improvements may include integrating real-time GPS tracking, adding payment gateway options, providing driver and passenger ratings, enabling live ride-tracking, and incorporating AI-based ride suggestions. These enhancements would further improve the system's reliability, attractiveness, and commercial viability.

14. References

Django Documentation, Python Language Guide, W3Schools HTML/CSS, Bootstrap Documentation, and other standard web development resources.