Car-Pooling-System (Backend API with Geo-Distance Calculation & JWT Authentication)

Brief Description :

The Car-Pooling-System is a backend API solution designed to facilitate ride-sharing or lift-sharing among users, specifically drivers and passengers.

The system aims to promote environmentally friendly travel by enabling users to share their journeys, consequently reducing carbon emissions and traffic congestion.

The Carpool System automatically calculates the distance between any two locations by generating their geographic coordinates from the provided start and end points.

Users can find matches for their rides through various mediums and communicate with each other to arrange trip details.

The Car-Pooling-System is a **functions based** backend APIs that uses **JWT authentication** and supports role-based activities (**Admin, Driver, Passenger, Visitors**).

Backend Tools & Frameworks: **Python 3.12.3, Django 4.2.23, Django REST Framework 3.16.6, MySQL Workbench**. Postman is used for API testing.

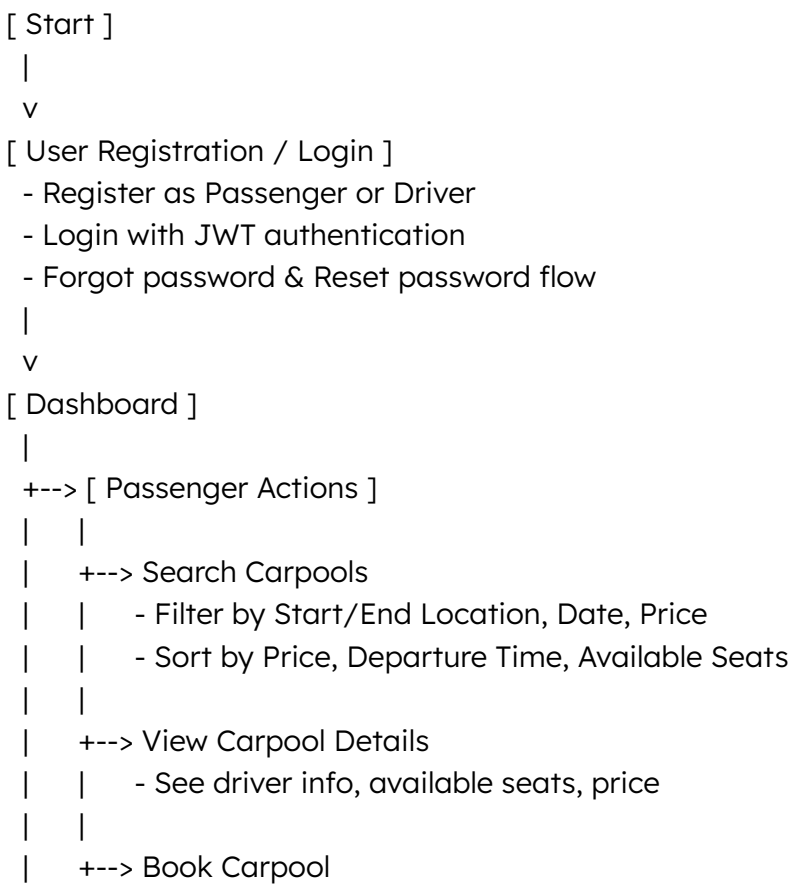
This document provides detailed information about all API endpoints in the Carpool Management System, including authentication, admin controls, carpool creation, booking.

Key Features:

- **JWT Authentication:** Secure login/logout system with role-based access (Admin, Driver, Passenger, Visitor).

- **Geo Distance:** Auto-calculates distance between any two locations using latitude & longitude (Haversine formula + Nominatim geocoding).
- **Nearby Rides:** Finds carpools within a 20 km radius dynamically.
- **Automated Notifications:** Sends booking updates and ride reminders via email.
- **Admin Insights:** Generate Overview reports, revenue analytics, busiest routes, and cancellation stats.
- **Custom Permissions:** Flexible access control for each role.
- **Searching, Filtering & Sorting:** Carpools and bookings can be searched/filtered/sorted by location, price, date, and seat availability.
- **Ride Start / End:** Drivers can now start and end rides, automatically updating ride status from upcoming → active → completed with validations for bookings and departure time.

Flow Diagram of Carpool System:



```
|     |     - Choose seats, confirm booking
|     |
|     +--> View My Bookings
|     |     - Update / Cancel bookings
|     |     - Filter by status, date, driver
|     |
|     +--> Receive Notifications
|           - Ride reminders 40 minutes before departure
|     |
|     +--> Dashboard
|
+--> [ Driver Actions ]
|     |
|     +--> Create Carpool
|     |     - Add route, seats, departure time, price
|     |
|     +--> Update / Delete Carpool
|     |
|     +--> View My Carpools
|     |     - See booked passengers
|     |
|     +--> Approve / Reject Bookings
|     |     - Confirm passengers for upcoming rides
|     |
|     +--> Start/End Ride
|     |     - Ride Start = True? >> Ride status -> 'Active' elif 'Upcoming'
|     |     - Ride End = True? >> Ride status -> 'Completed' elif  'Active'
|     |
|     +--> Notifications
|           - Receive booking updates
|
+--> [ Admin Actions ]
      |
      +--> View All Users
      |     - Activate / Deactivate accounts
      |
      +--> View All Activities
      |     - Track user activity logs
      |
      +--> Manage Carpools & Bookings
      |     - View all carpools, bookings
```

```
        |
        +--> Admin Dashboard
        |     - User Carpools, Bookings, earnings reports, busiest routes, cancellations
```

Passenger Workflow (Detailed):

```
[ Login/Register ]
 |
 v
[ Dashboard ]
 |
 +--> Search Carpools
 |     |
 |     +--> View Available Carpools
 |     |
 |     +--> Apply Filters (Location, Date, Price)
 |
 +--> Book Carpool
       |
 +--> Seats Confirmation
       +--> Booking Confirmation
       |
       +--> Receive Email Notification
```

Driver Workflow (Detailed):

```
[ Login/Register ]
 |
 v
[ Dashboard ]
 |
 +--> Create Carpool
 |     - Add Car Details, Departure Time, Seats, Price
 |
 +--> View My Carpools
 |     - See passenger bookings
 |
 +--> Approve / Reject Bookings
 |     - Update booking status
 |
 +--> Update / Delete Carpools
```

|
+--> Start Ride (Upcoming → Active)
|
+--> End Ride (Active → Completed)
|
+--> Receive Booking Notifications

Admin Workflow (Detailed):

[Login]
|
v
[Admin Dashboard]
|
+--> Manage Users
| - View all users
| - Activate / Deactivate accounts
|
+--> Manage Carpools & Bookings
| - View all carpools
| - View all bookings
|
+--> Reports
| - Revenue by route
| - Weekly / Monthly bookings
| - Cancellation stats

Summary of Entities and Actions:

[Users (Roles)]
- Passenger
- Driver
- Admin
- Actions: Register, Login, Update Profile, Delete Profile

[Carpools]
- Created by Drivers
- Viewed by Passengers
- Actions: Create, Update, Delete, View, Search, Sort

[Bookings]
- Made by Passengers
- Approved/Rejected by Drivers
- Actions: Create, Update, Cancel, Filter

[Notifications]
- Sent to Passengers and Drivers
- Actions: Ride reminders, Booking updates

[Admin Reports]
- View All Users
- View All Activities
- View All Carpools and Bookings
- Manage Search data
- Generate Reports

Custom Authentication & Permissions:

Class: CustomJWTAuthentication

Functionality: Validates JWT token and retrieves User.

Raises **AuthenticationFailed** for invalid or expired tokens.

Permission Classes:

- **IsAuthenticatedCustom** → Any authenticated user
- **IsAdminCustom** → Admin only
- **IsDriverCustom** → Driver only
- **IsPassengerCustom** → Passenger only
- **IsAdminOrDriverCustom** → Admin or Driver
- **IsAdminOrPassengerCustom** → Admin or Passenger
- **IsDriverOrPassengerCustom** → Driver or Passenger

Functions List

Total Functions: 33 divided into Authentication, User Management, Carpool Management, Booking Management, Admin Reporting, Notifications.

Each function is **explained like a docstring**, with **endpoint, method, authentication, description, and request body example**.

★ Authentication & User Management Functions:

1. register_user()

"""

Register a new user with username, email, and password.
This will create a passenger account in the system.

"""

Method: POST

Endpoint: /register/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "username": "vills",
  "email": "vills@example.com",
  "password": "StrongPass@123",
  "first_name": "Vills",
  "last_name": "Patel",
  "profile_image": "profile.jpg"
}
```

2. login_user()

"""

Login a registered user using username and password.
Generates JWT token for session management.

"""

Method: POST

Endpoint: /login/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "username": "vills",
  "password": "StrongPass@123"
}
```

3. logout_user()

"""

Logout the currently logged-in user by blocklisting the JWT access token.

"""

Method: POST

Endpoint: /logout_user/

Authentication: Required

Request Body: None

4. view_profile()

"""

View the profile details of the logged-in user.
Returns first name, last name, email, username, profile image.

"""

Method: GET

Endpoint: /profile/my-profile/

Authentication: ✅ Required

Request Body: None

5. update_profile()

"""

Update the current user's profile information such as username, email, password, name, or profile image.

"""

Method: PUT

Endpoint: /profile/update-profile/

Authentication: ✅ Required

Request Body Example:

```
{
  "username": "newvills",
  "email": "newvills@example.com",
  "password": "Updated@123",
  "first_name": "Vills",
  "last_name": "Updated",
  "profile_image": "new_profile.jpg"
}
```

6. user_dashboard()

"""

Fetch dynamic user dashboard data including overview, user info, user's carpools with extra details, and bookings with extra info.

“”"

Method: GET

Endpoint: profile/user-dashboard/

Authentication: ✅ Required

Request Body Example: None

7. delete_profile()

"""

Delete the currently logged-in user's account permanently.

"""

Method: DELETE

Endpoint: /profile/delete-profile/

Authentication: ✅ Required

Request Body Example:

```
{
  "user_id": 12
}
```

8. forgot_password()

"""
Initiate password reset by sending OTP to registered email.
"""

Method: POST

Endpoint: /profile/forgot-password/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "email": "vills@example.com"
}
```

9. reset_password()

"""
Reset user password using OTP received via email.
"""

Method: POST

Endpoint: /profile/reset-password/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "email": "vills@example.com",
  "password": "NewPass@123",
  "OTP": "458921"
}
```

10. contact_us()

"""
Send a message to the support team from logged-in or guest users.
"""

Method: POST

Endpoint: /profile/contact-us/

Authentication: ❌ Optional

Request Body Example:

```
{
  "email": "vills@example.com",
  "message": "I need help with my booking."
}
```

★ Carpool Management Functions

11. create_carpool()

"""
Create a new carpool as a driver.
Provide departure time, seats, price, and route details.
When publishing a carpool journey, the system automatically calculates the distance between start and end locations if the user does not provide it.
"""

Method: POST

Endpoint: /carpool/create/

Authentication: ✅ Required

Request Body Example:

```
{
  "car_name": "Toyota Prius",
  "departure_time": "2025-09-25T08:30:00",
  "available_seats": 3,
  "price_per_seat": 200,
  "start_location": "Mumbai",
  "end_location": "Pune"
}
```

12. update_carpool()

"""
Update existing carpool information.
Only the carpool driver can update the carpool.
""""

Method: PUT

Endpoint: /carpool/update/

Authentication: ✅ Required

Request Body Example:

```
{
  "carpool_id": 5,
  "available_seats": 2,
  "price_per_seat": 250
}
```

13. delete_carpool()

"""

Delete a carpool created by the driver.

"""

Method: DELETE

Endpoint: /carpool/delete/

Authentication: ✅ Required

Request Body Example:

```
{
  "carpool_id": 5
}
```

14. view_my_carpools()

"""

View all carpools created by the logged-in driver.

"""

Method: GET

Endpoint: /carpool/my-carpools/

Authentication: ✅ Required

Request Body: None

15. view_booked_passenger()

"""

View passengers who booked a particular carpool.

"""

Method: GET

Endpoint: /carpool/view-passengers/

Authentication: ✅ Required (Driver only)

Request Body Example:

```
{
  "carpool_id": 5
}
```

16. carpool_detail()

"""

Public endpoint to view details of upcoming carpools.
Only shows carpools with available seats.

"""

Method: GET

Endpoint: /carpool/detail/

Authentication: ❌ Not Required

Request Body: None

17. search_carpools()

"""

Search carpools by start location, end location, or date.
Nearby Rides in radius of 20 km

"""

Method: POST

Endpoint: /carpool/search-carpools/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "start_location": "Mumbai",
  "end_location": "Pune",
  "date": "2025-09-25"
}
```

18. sort_carpools_by()

"""
Sort available carpools by departure time, price, or available seats.
If the ride is completed successfully, it will return a 200 status code with a message "Ride completed successfully."
If the ride is cancelled due to arrival time being passed, it will return a 400 status code with a message "Cannot start ride, arrival time passed. Ride cancelled.
"""

Method: POST

Endpoint: /carpool/sort-carpools/

Authentication: ❌ Not Required

Request Body Example:

```
{
  "sort_by": "price",
  "order": "asc"
}
```

19. start_end_ride_driver()

"""
This API is used to start or end a ride as a driver.
"""

Method: POST

Endpoint: /carpool/start-end-rides/

Authentication: ✅ Required

Request Body Example:

```
{
  “carpool_id”: “121”,
  “start_ride”: “True”,
```

“end_ride”: “True”
}

★ Booking Management Functions

20. book_carpool()

"""
Book a seat in an upcoming carpool.
Only registered users can book.
While booking a carpool, the system now calculates the passenger's travel distance between their selected start and end locations automatically.
"""

Method: POST

Endpoint: /booking/create/

Authentication: ✅ Required

Request Body Example:

{
"carpool_id": 5,
"seats_booked": 1
}

21. my_bookings_info()

"""
View all bookings made by the logged-in user.
"""

Method: GET

Endpoint: /booking/my-bookings/

Authentication: ✅ Required

Request Body: None

22. update_my_booking()

"""

Update number of seats in an existing booking.
"""

Method: PUT

Endpoint: /booking/update/

Authentication: ✅ Required

Request Body Example:

```
{
  "booking_id": 10,
  "seats_booked": 2
}
```

23. cancel_booking()

"""
Cancel an existing booking.
"""

Method: DELETE

Endpoint: /booking/delete/

Authentication: ✅ Required

Request Body Example:

```
{
  "booking_id": 10
}
```

24. filter_bookings()

"""
Filter user bookings by date, status, or driver.
"""

Method: POST

Endpoint: /booking/filter/

Authentication: ✅ Required

Request Body Example:

```
{
  "status": "confirmed",
  "date": "2025-09-25"
}
```

25. driver_view_booking_requests()

"""

Drivers can view all pending booking requests for their carpools.

"""

Method: GET

Endpoint: /driver/booking-requests/

Authentication: ✅ Required (Driver only)

Request Body: None

26. driver_approve_reject_booking()

"""

Drivers can approve or reject a booking request.

"""

Method: POST

Endpoint: /driver/booking-action/

Authentication: ✅ Required (Driver only)

Request Body Example:

```
{
  "booking_id": 10,
  "action": "approve"
}
```

27. give_review_rating()

"""

Add review for a driver

"""

Method: POST

Endpoint: /review/

Authentication: ✅ Required (Driver/Passanger only)

Request Body Example:

```
{
 "review_given_to_name": "mitesh@1"
  "carpool_id": 10,
  "booking_id": "20"
 "rating": "5"
 "comment": "Nice ride!"
}
```

28. view_driver_info()

"""
View basic details of carpool's Driver for passenger
"""

Method: GET

Endpoint: /driver-info/

Authentication: ❌ Not Required

Request Body Example:

```
{
 "driver_user_id": "mitesh@1"
 }
```

★ Notification Function

29. ride_reminder_notifications()

"""
Send email reminders to passengers 40 minutes before the ride.
Returns "No upcoming rides" if none.
"""

Method: GET

Endpoint: /ride_reminder_notifications/

Authentication: ✅ Required (Driver only)

Request Body: None

★ Admin Functions

30. admin_view_users()

"""
List all registered users in the system.
Filters by username/email are optional.
"""

Method: POST

Endpoint: /admin/view-users/

Authentication: ✅ Required (Admin/Superuser)

Request Body:

```
{
  "username": "john",      # string (optional)
  "email": "john@example.com"  # string (optional)
}
```

31. admin_active_deactive_user()

"""
Activate or deactivate a user account.
"""

Method: POST

Endpoint: /admin/user-action/

Authentication: ✅ Required (Admin/Superuser)

Request Body:

```
{
  "user_id": 12,          # int
  "action": "deactivate"   # string ("activate" or "deactivate")
}
```

32. view_all_activities()

"""
Retrieve activity logs of all users in the system.
Filters optional: by user or action type.
"""

Method: POST

Endpoint: /admin/view-activities/

Authentication: ✅ Required (Admin/Superuser)

Request Body:

```
{
  "user_id": null,         # int (optional)
  "action_type": "booking" # string (optional)
}
```

33. admin_view_carpools()

"""
View all carpools including upcoming and past journeys.
Filters optional: by departure/arrival location or date.
"""

Method: POST

Endpoint: /admin/carpools/

Authentication: ✅ Required (Admin/Superuser)

Request Body:

```
{
  "departure_location": "Ahmedabad", # string (optional)
  "arrival_location": "Mumbai",      # string (optional)
  "date": "2025-09-25"               # string (optional)
}
```

34. admin_view_bookings()

"""
View all carpool bookings: pending, confirmed, or rejected.
Filters optional: by carpool_id, passenger_name, or status.
"""

Method: POST

Endpoint: /admin/carpool-bookings/

Authentication: ✅ Required (Admin/Superuser)

Request Body:

```
{
  "carpool_id": 1,          # int (optional)
  "passenger_name": "John",  # string (optional)
  "status": "confirmed"      # string (optional: "pending", "rejected", "confirmed")
}
```

35. admin_full_report()

"""
Display Dashboard of overall Carpool system report.
Includes users, carpools, bookings, cancellations, and revenue.
"""

Method: POST

Endpoint: /admin/reports/

Authentication: ✅ Required (Admin/Superuser)

Request Body: None