CARPOOL RIDE SHARING APP

Report

COEN275

Object-Oriented Analysis, Design and Programming

Prepared by
Aparnaa Pazhanivelan(07700011640)
Pujitha Kallu(W1653660)
Sowmya Baluvu(07700011659)

Under the guidance of Prof. Paddu Melanahalli

Contents

Project Statement:

A ride-sharing platform that connects drivers and passengers traveling the same routes, helping reduce costs and carbon emissions.

Objective:

- Provide a seamless platform for carpooling and ride-sharing, connecting drivers and passengers.
- The project is a web-based carpooling/ride-sharing service application.
- It uses Google Maps API for location-based services, route planning, and mapping functionality.
- The backend is Java-based, using Spring Boot framework.
- The application allows users to register, create profiles, and manage their account information.
- Users offer rides by specifying their route, available seats, and departure time.
- The system includes a ride-matching algorithm to connect drivers with potential passengers.
- A booking system for users to request and confirm rides.
- The project structure is a separation of concerns between backend and frontend components.
- The application include features like ride history, recurring rides, past rides
- Allow users to search and filter rides based on preferences like timing, routes.
- Generates payment details for one time rides and recurring rides.

Overview:

Project Features:

1. Global Exception Handling: Implements a centralized exception-handling mechanism for the entire API.

2. Java: The primary programming language used for the project.

3. Spring Boot: A popular framework for building Java-based enterprise applications, particularly well-suited for creating standalone, production-grade Spring-based Applications.

4. Spring MVC: Used for building the web layer of the application, handling HTTP requests and responses.

5. Spring's Exception Handling: Utilizing `@ControllerAdvice` and `@ExceptionHandler` for global exception handling.

6. RESTful API Design: The application is structured to follow REST principles for API design.

7. Maven: These build tools are used for dependency management and project building (based on the standard Spring Boot project structure).

8. HTTP Status Codes: Proper use of HTTP status codes for different scenarios (e.g., 404 for Not Found, 500 for Internal Server Error).

9. DTOs (Data Transfer Objects): Used for structuring the data sent in API responses, particularly for error messages.

10. Database Integration: Spring Data JPA for database operations.

11. Dependency Injection: Spring's core feature, used for managing application components.

12. Unit and Integration Testing: Using JUnit and Spring's testing support.

13. Postman: A collaboration platform and API testing tool used to build, test, and document APIs efficiently.

14. Map Integration: Use Google Maps APIs for geolocation and route management.

15. Ride Matching Algorithm: Add logic to match passengers with rides based on location and timing.

16. Ride History: Provide users with a history of rides they’ve taken.

Project Requirements:

1.User Profiles: Both drivers and passengers create profiles with their travel preferences and routes.

2. Route Matching: The system matches passengers with drivers heading to the same destination at a similar time.

3. Fare Splitting: Automatically calculate and split fares between passengers, with secure in-app payments.

4. Ride Scheduling: Users can schedule future rides or set up recurring carpooling for daily commutes.

Key Features:

1.User Management:

- Handles user profiles, including name, email, phone number, and role (Driver or Passenger).
- Users can own vehicles and participate in rides.

2.Vehicle Management:

- Allows users to register their vehicles with details like number, type, color, and seat count.
- Vehicles are associated with users and linked to rides.

3. Ride Management:

- Users (Drivers) can create rides, specifying pickup and destination locations, start and end times, and available seats.
- Rides are associated with vehicles and participants.

4. Location Management:

- Handles locations with latitude, longitude, and address.
- Pickup and destination locations are linked to rides.

5. Ride Participation:

- Manages the participation of users in rides, identifying their roles (Driver or Passenger).
- Tracks each participant's association with rides.

6. Role Management:

- Differentiates between drivers and passengers.
- Ensures that each ride has at least one driver.

7. Search:

- Provide users with the ability to search and filter rides based on:
- Pickup and drop-off locations.

8. Ride History:

- Allow users to view their past rides, including dates, routes, and payment history.
- Maintain a dedicated table in the database for storing completed rides.

9. Recurring Rides:

- Enable users to schedule recurring rides (e.g., daily commute to work).

10. Ride Creation:

- Drivers can create rides with details like pickup and destination locations, available seats, and timing.

11. Joining Rides:

- Passengers can join rides as participants, reducing travel costs and promoting eco-friendly transport.

System Architecture:

System architecture refers to the conceptual model that defines the structure and behavior of a system. It includes layers such as the Presentation Layer (Frontend), Application Layer (Backend), Data Layer, Database, Security Layer, External Services, and Configuration and Deployment. Key design principles include separation of concerns, scalability, maintainability, and security. In the carpooling application, the architecture is based on the **Model-View-Controller (MVC)** design pattern, which ensures a clear separation between the data model, user interface, and business logic. This approach enhances maintainability and scalability while facilitating efficient ride management, user authentication, payment processing

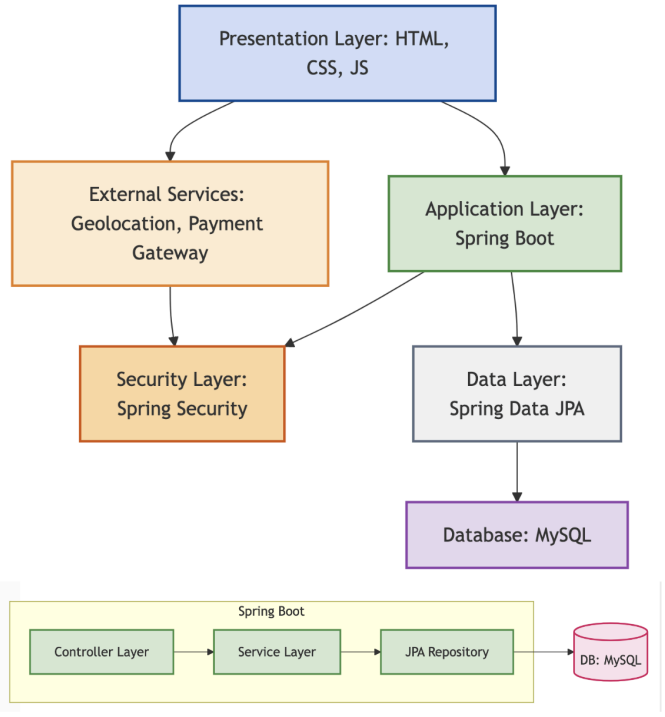


Fig1: System Architecture

Technology Stack:

Backend:

- **Java + Spring Boot**:
    - Handles business logic, RESTful APIs, and database interactions.
    - Frameworks used: Spring Data JPA, Spring MVC.
- **Database**:
    - MySQL for storing and managing relational data.

Testing:

- Postman for API testing.
- Unit tests for service and repository layers.
- Integration tests

Frontend:

JS/HTML

The carpooling application utilizes JavaScript and HTML for its frontend, integrating the Google Maps API for interactive map functionality. Users can dynamically set their start and destination locations by clicking on the map, which updates latitude, longitude, and address fields in real-time using geocoding. The application also provides reverse geocoding to translate coordinates into human-readable addresses. The modular and scalable implementation enhances usability, with potential for future integration of features like route visualization and real-time navigation using the Directions API.

Styling:

- **CSS Frameworks:**
    - Bootstrap: For responsive design and prebuilt components.
    - For utility-first CSS styling.
- **Custom CSS/SCSS:**
    - For branding and unique UI/UX requirements.

Authentication:

- Handle login/logout functionality.
- Use **JWT** (JSON Web Tokens) to securely manage authentication and user sessions.

Tables:

1.User:

```
mysql> SHOW COLUMNS FROM carpool.User;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | bigint       | NO   | PRI | NULL    | auto_increment |
| dob          | datetime(6)  | YES  |     | NULL    |                |
| emailId      | varchar(255) | YES  |     | NULL    |                |
| firstName    | varchar(255) | YES  |     | NULL    |                |
| lastName     | varchar(255) | YES  |     | NULL    |                |
| password     | varchar(255) | YES  |     | NULL    |                |
| phoneNumber  | varchar(255) | YES  |     | NULL    |                |
| profileImage | longblob     | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)

mysql>
```

Fig2: UserTable

2.Vehicle:

```
mysql> SHOW COLUMNS FROM carpool.Vehicle;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | bigint       | NO   | PRI | NULL    | auto_increment |
| color     | varchar(255) | YES  |     | NULL    |                |
| name      | varchar(255) | YES  |     | NULL    |                |
| number    | varchar(255) | YES  |     | NULL    |                |
| seatCount | int          | NO   |     | NULL    |                |
| type      | varchar(255) | YES  |     | NULL    |                |
| ownerId   | bigint       | YES  | MUL | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql>
```

Fig3: Vehicle Table

3.Ride:

```
mysql> SHOW COLUMNS FROM carpool.Ride;
+----------------------+-----------------------------------------------+------+-----+---------+----------------+
| Field                | Type                                          | Null | Key | Default | Extra          |
+----------------------+-----------------------------------------------+------+-----+---------+----------------+
| id                   | bigint                                        | NO   | PRI | NULL    | auto_increment |
| availableSeats       | int                                           | NO   |     | NULL    |                |
| createdDate          | date                                          | YES  |     | NULL    |                |
| date                 | date                                          | YES  |     | NULL    |                |
| daysOfWeek           | varchar(255)                                  | YES  |     | NULL    |                |
| endTime              | time(6)                                       | YES  |     | NULL    |                |
| startTime            | time(6)                                       | YES  |     | NULL    |                |
| status               | enum('ACTIVE','CANCELLED','COMPLETED','CREATED') | YES  |     | NULL    |                |
| type                 | enum('ONE_TIME','RECURRING')                  | YES  |     | NULL    |                |
| destinationLocationId | bigint                                        | YES  | MUL | NULL    |                |
| pickupLocationId     | bigint                                        | YES  | MUL | NULL    |                |
| vehicleId            | bigint                                        | YES  | MUL | NULL    |                |
+----------------------+-----------------------------------------------+------+-----+---------+----------------+
12 rows in set (0.00 sec)
```

Fig4: RideTable

4.Ride Participant:

```
mysql> SHOW COLUMNS FROM carpool.RideParticipant;
+-------------+----------------------------+------+-----+---------+----------------+
| Field       | Type                       | Null | Key | Default | Extra          |
+-------------+----------------------------+------+-----+---------+----------------+
| id          | bigint                     | NO   | PRI | NULL    | auto_increment |
| cancelledAt | date                       | YES  |     | NULL    |                |
| joinedAt    | date                       | YES  |     | NULL    |                |
| role        | tinyint                    | YES  |     | NULL    |                |
| status      | enum('ACTIVE','CANCELLED') | YES  |     | NULL    |                |
| userId      | bigint                     | YES  | MUL | NULL    |                |
| rideId      | bigint                     | YES  | MUL | NULL    |                |
+-------------+----------------------------+------+-----+---------+----------------+
7 rows in set (0.01 sec)
```

Fig5: Ride Participant Table

5.Location:

```
mysql> SHOW COLUMNS FROM carpool.Location;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | bigint       | NO   | PRI | NULL    | auto_increment |
| address   | varchar(255) | YES  |     | NULL    |                |
| latitude  | double       | NO   |     | NULL    |                |
| longitude | double       | NO   |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Fig6: Location Table

6.Fare:

```
mysql> SHOW COLUMNS FROM carpool.Fare;
+--------+---------+------+-----+---------+----------------+
| Field  | Type    | Null | Key | Default | Extra          |
+--------+---------+------+-----+---------+----------------+
| id     | bigint  | NO   | PRI | NULL    | auto_increment |
| amount | double  | NO   |     | NULL    |                |
| status | tinyint | YES  |     | NULL    |                |
| rideId | bigint  | YES  | UNI | NULL    |                |
+--------+---------+------+-----+---------+----------------+
4 rows in set (0.03 sec)
```

Fig7: Fare Table

7.Wallet:

```
mysql> SHOW COLUMNS FROM carpool.Wallet;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id             | bigint      | NO   | PRI | NULL    | auto_increment |
| balance        | double      | YES  |     | NULL    |                |
| billingAddress | varchar(255)| YES  |     | NULL    |                |
| cardHolderName | varchar(255)| YES  |     | NULL    |                |
| cardNumber     | varchar(255)| YES  |     | NULL    |                |
| expiryDate     | varchar(255)| YES  |     | NULL    |                |
| userId         | bigint      | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)

mysql>
```

Fig8: Wallet Table

8.RecurringToOneTimeRideLink:

```
mysql> SHOW COLUMNS FROM carpool.RecurringToOneTimeRideLink;
+-----------------+--------+------+-----+---------+----------------+
| Field           | Type   | Null | Key | Default | Extra          |
+-----------------+--------+------+-----+---------+----------------+
| id              | bigint | NO   | PRI | NULL    | auto_increment |
| oneTimeRideId   | bigint | YES  | UNI | NULL    |                |
| recurringRideId | bigint | YES  | MUL | NULL    |                |
+-----------------+--------+------+-----+---------+----------------+
3 rows in set (0.01 sec)

mysql>
```

Fig9: RecurringToOneTimeRideLink Table

9. Transaction:

```
mysql> SHOW COLUMNS FROM carpool.Transaction;;
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| id            | bigint      | NO   | PRI | NULL    | auto_increment |
| amount        | double      | NO   |     | NULL    |                |
| completedDate | date        | YES  |     | NULL    |                |
| description   | varchar(255)| YES  |     | NULL    |                |
| status        | tinyint     | YES  |     | NULL    |                |
| type          | tinyint     | YES  |     | NULL    |                |
| fareId        | bigint      | YES  | MUL | NULL    |                |
| userId        | bigint      | YES  | MUL | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Fig10: Transaction Table

10. RideParticipant:

```
mysql> SHOW COLUMNS FROM carpool.RideParticipant;;
+-------------+----------------------------+------+-----+---------+----------------+
| Field       | Type                       | Null | Key | Default | Extra          |
+-------------+----------------------------+------+-----+---------+----------------+
| id          | bigint                     | NO   | PRI | NULL    | auto_increment |
| cancelledAt | date                       | YES  |     | NULL    |                |
| joinedAt    | date                       | YES  |     | NULL    |                |
| role        | tinyint                    | YES  |     | NULL    |                |
| status      | enum('ACTIVE','CANCELLED') | YES  |     | NULL    |                |
| userId      | bigint                     | YES  | MUL | NULL    |                |
| rideId      | bigint                     | YES  | MUL | NULL    |                |
+-------------+----------------------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

Fig10: RideParticipant Table

UML Diagram:

This UML diagram represents a carpooling system, detailing key entities like User, Vehicle, Ride, Location, RideParticipant, and Fare. It illustrates how users can own vehicles, create or join rides, and manage their roles as drivers or passengers. The system supports ride management with pickup and destination locations, fare calculation, and role-based participation. Enumerations like RideStatus define the state of rides and payments, ensuring a structured and efficient process for booking and sharing rides.

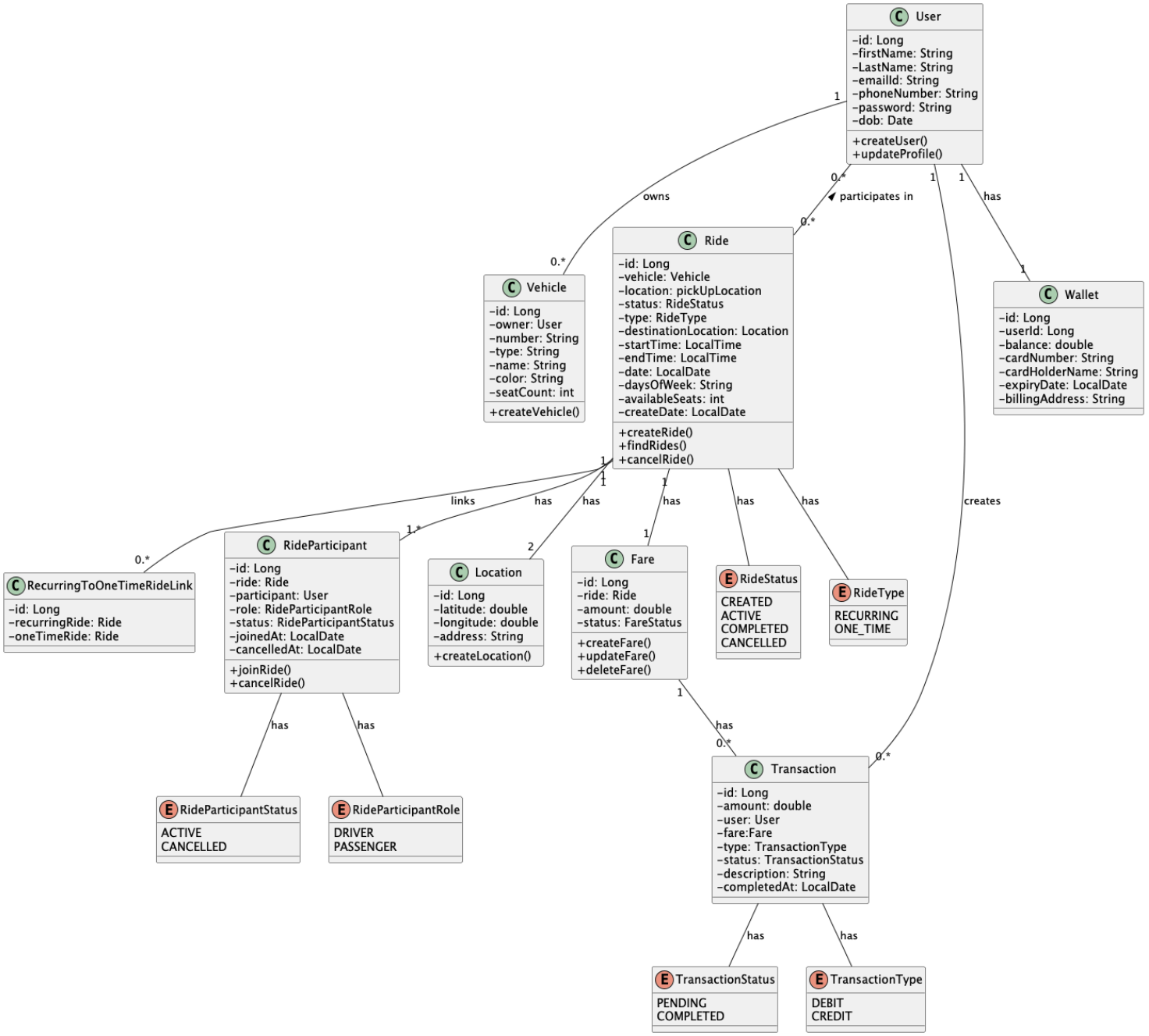


Fig11: UML Diagram

CLASS Diagram:

The diagram represents a carpooling system where a user can own multiple vehicles and participate in multiple rides. Each vehicle belongs to a specific user and includes details such as number, type, and seat count. Rides are created by users, linked to a vehicle, and associated with specific locations for pickup and drop-off. Rides also include details like start time, end time, available seats, and ride status. Ride participants are users who join rides, either as drivers or passengers, with roles and statuses tracked for each participant. Locations are defined by latitude, longitude, and address, representing the start and destination points of a ride. Fares are tied to specific rides, including the amount and status, such as pending or completed. The relationships between these entities

define how users interact with vehicles, rides, locations, and payments in the carpooling system.

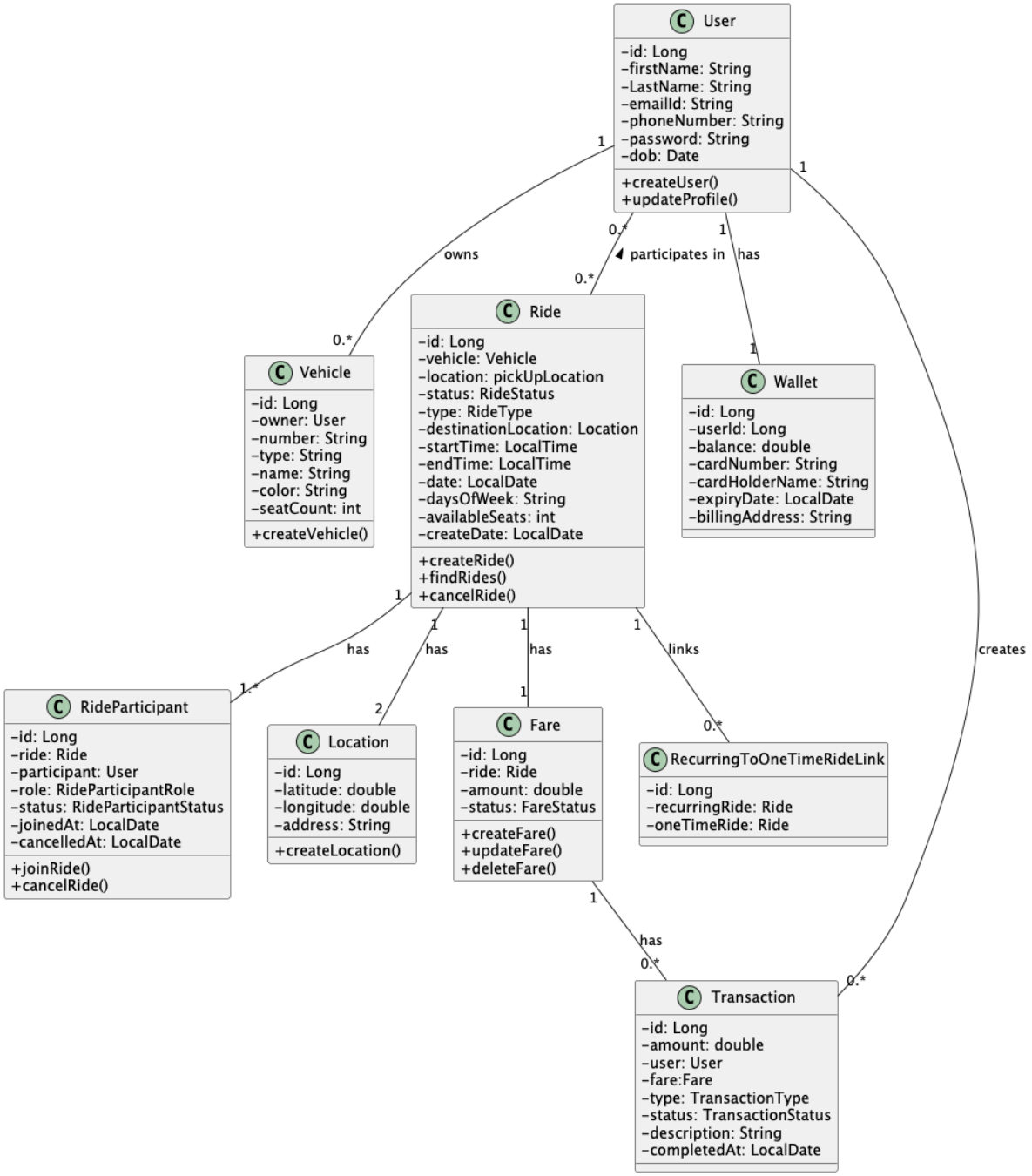


Fig12: Class Diagram

Object oriented concepts used in the application

1. Classes and Interfaces:

- **Classes**: Represent entities such as User, Vehicle, Ride, RideParticipant, Location, and Fare. These classes encapsulate attributes and behaviors, serving as blueprints for creating objects in the system.
- **Interfaces**: Used to define contracts for service layers like UserService, RideService, and VehicleService, ensuring consistent implementation across the application.

Usage: Classes like UserController, RideController, VehicleController, and RideService encapsulate functionality and attributes related to user management, ride management, and vehicle operations.

2. Attributes and Methods:

- **Attributes**: Represent the state of the class (e.g., name, email, vehicleNumber, seatCount, etc.).
- **Methods**: Define the behavior of classes (e.g., createRide, joinRide, updateUserProfile).

Usage:

- Users have attributes like name, email, and phoneNumber, and methods like createUser and updateProfile.
- Ride includes attributes such as startTime, endTime, and availableSeats, and methods like createRide and cancelRide.

3. Inheritance:

- **Definition**: Allows a subclass to inherit attributes and methods from a superclass.
- **Usage**:
    - Driver and Passenger can inherit from User.
    - Abstract classes or interfaces like BaseService may define common functionality that is inherited by RideService, UserService, etc.

4. Composition:

- **Definition**: A relationship where one class is composed of other objects.
- **Usage**:
    - Ride has a Vehicle and Location (pickup and destination).
    - RideParticipant has a User (passenger or driver) and a Ride.

5. Association:

- **Definition**: Represents relationships between classes (one-to-one, one-to-many, many-to-many).
- **Usage**:
    - Users have an association with Vehicle (a user can own multiple vehicles).
    - Ride is associated with Location (pickup and destination) and RideParticipant (drivers and passengers).
    - RideParticipant has an association with Ride (each participant is part of a ride).

6. Aggregation:

- **Definition**: A specific type of association that shows a "whole-part" relationship.
- **Usage**:

- Ride aggregates RideParticipant objects (a ride consists of multiple participants).
- User aggregates Vehicle objects (a user owns multiple vehicles).

7. Dependency Injection:

- **Definition**: Dependencies are injected externally, promoting loose coupling and testability.
- **Usage**:
  - *UserService* depends on *UserRepository* and is injected via Spring's @Autowired annotation.
  - *RideService* depends on *RideRepository*, *VehicleRepository*, and *LocationRepository*.

Design Pattern:

1. Model (Backend):

- The User, Vehicle, Ride, Location, RideParticipant, and Fare classes in the backend represent the model layer.
- These classes encapsulate the data and business logic, defining the core entities of the system.

2. Controller (Backend):

- The UserController, VehicleController, RideController, RideParticipantController, and FareController classes act as controllers.
- They handle incoming HTTP requests from the frontend, interact with the service layer, and return appropriate responses (e.g., JSON data).
- Examples:
  - UserController: Manages user-related operations like registration and profile updates.
  - RideController: Handles ride creation, retrieval, and updates.

3. Service Layer Pattern:

- The application follows a service layer pattern where business logic is encapsulated in service classes (UserService, RideService, etc.).
- These services provide clean interfaces for controllers to interact with the business logic and ensure separation of concerns.
- Benefits:
  - Improves code modularity, making it easier to maintain and extend.
  - Facilitates unit testing by isolating business logic from controllers.
- Example:
  - RideService: Handles business rules for ride creation, availability checks, and ride updates.

4. JPA Repository Pattern:

- The application leverages Java Persistence API (JPA) repository pattern for data persistence.
- Repository interfaces (UserRepository, RideRepository, VehicleRepository, etc.) interact with the MySQL database, allowing for CRUD operations and custom queries.
- Benefits:
  - Simplifies database access using object-oriented approaches.
  - Enhances efficiency and consistency with prebuilt methods like findById() and save().

5. DTO (Data Transfer Object) Pattern:

- DTOs (UserDto, RideDto, VehicleDto, etc.) are used to transfer data between the service layer and the controller layer.

- These objects ensure only required fields are exposed to the API consumers, improving security and performance.
- Benefits:
    - Reduces the number of method calls between layers.
    - Provides a flexible and version-friendly API design.

6. Communication Pattern:

- RESTful APIs: The frontend communicates with the backend through REST APIs, exchanging data in JSON format.
- Example:
    - The RideController handles a POST request to create a ride, processes the data using RideService, and persists it using RideRepository.

7. Distributed MVC Implementation:

- The system follows a distributed implementation of the Model-View-Controller (MVC) pattern:
    - Model: Defined by entity classes (User, Ride, etc.) and managed by JPA repositories.
    - Controller: Spring Boot controllers handle HTTP requests and provide endpoints.
    - View: The frontend (HTML/Css) handles the user interface and interacts with the backend via REST APIs.

Output Screenshots:

1.Home page: [Login Page for Existing User & Signup button for new User]

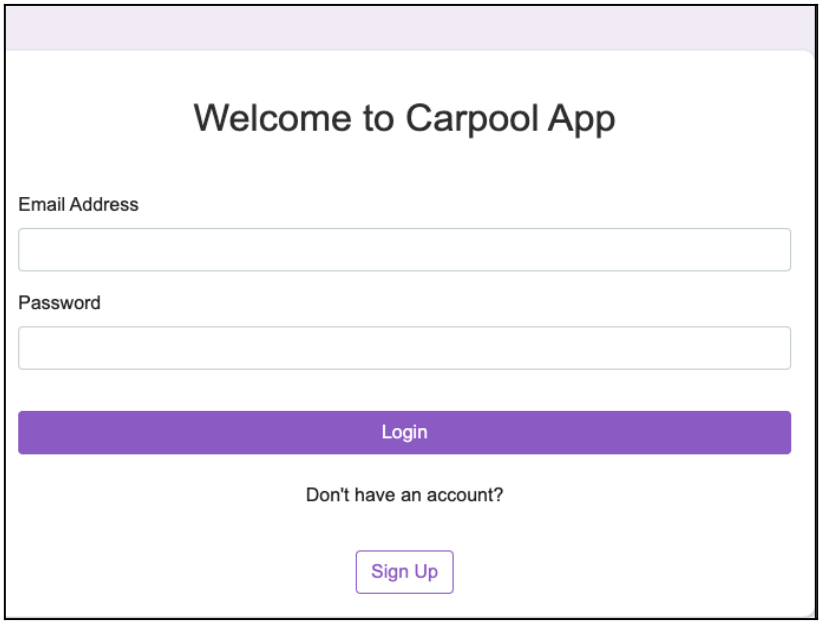


Fig13: HomePage

2.Signup page: [Signup details for new user]

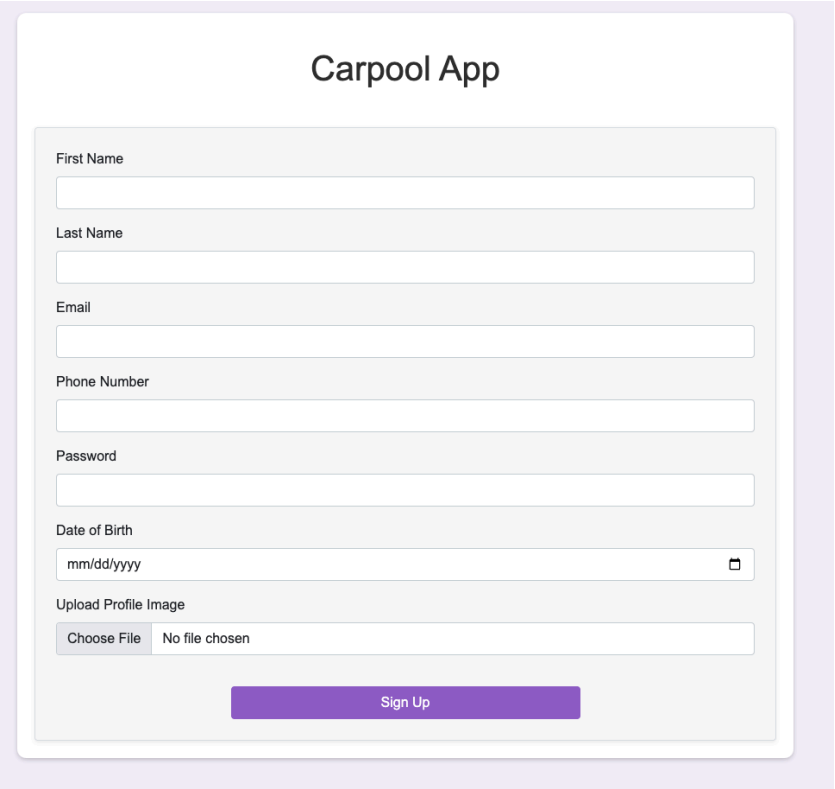


Fig14: Signup Page

3.User Dashboard: [HomeScreen where All the tabs are visible to user to perform actions]

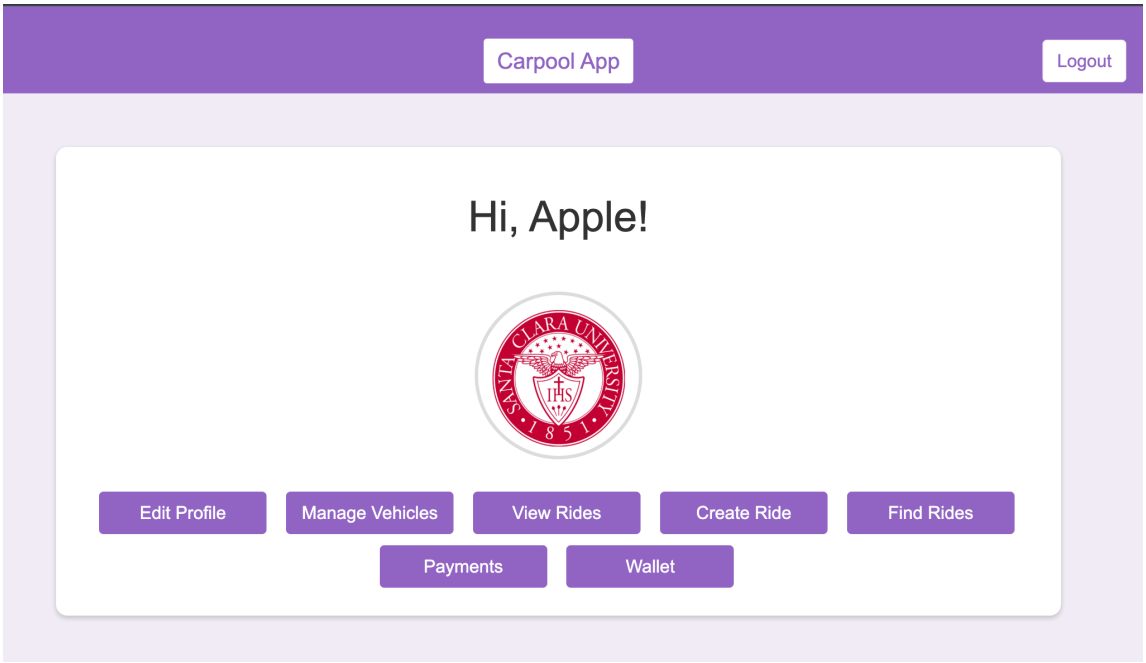


Fig15: User Dashboard

4.Edit Profile: [Click on Update for the Changes for UserProfile]

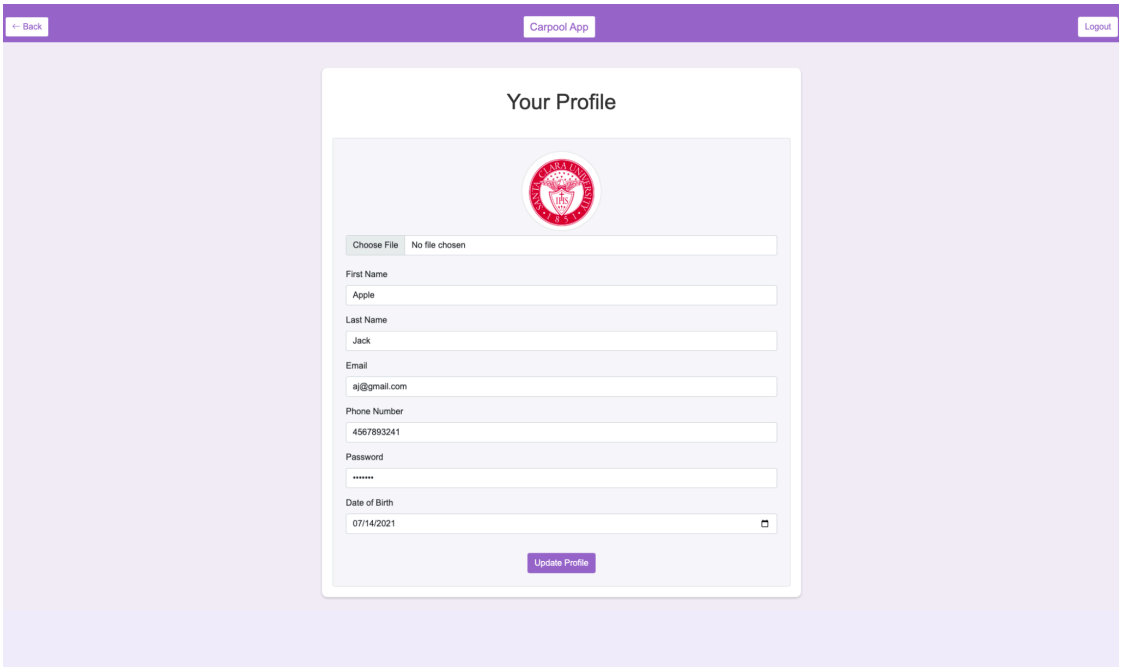


Fig16:EditProfile

5.Manage vehicles: [Showing Vehicle details with seat count]

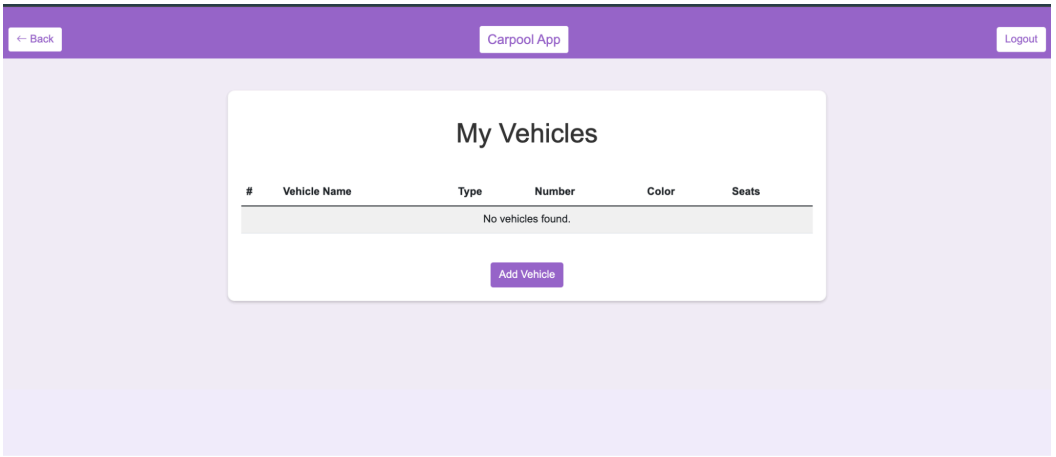


Fig17: Manage Vehicles[1st screen]

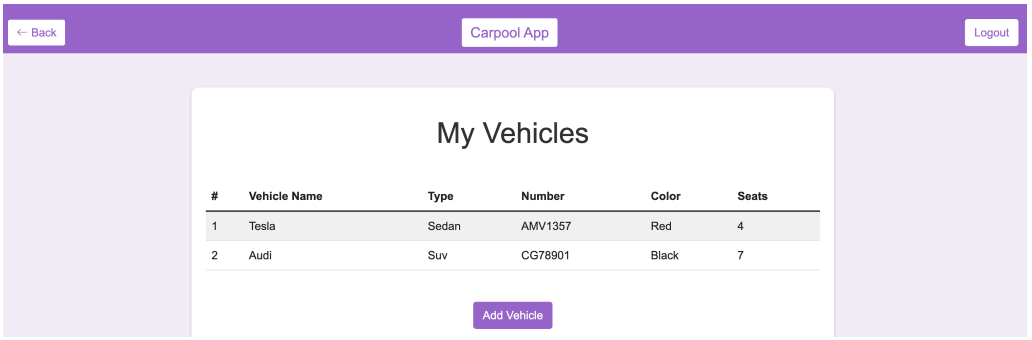


Fig18: Manage Vehicles[Vehicle Details]

- **Adding New Vehicle details:**

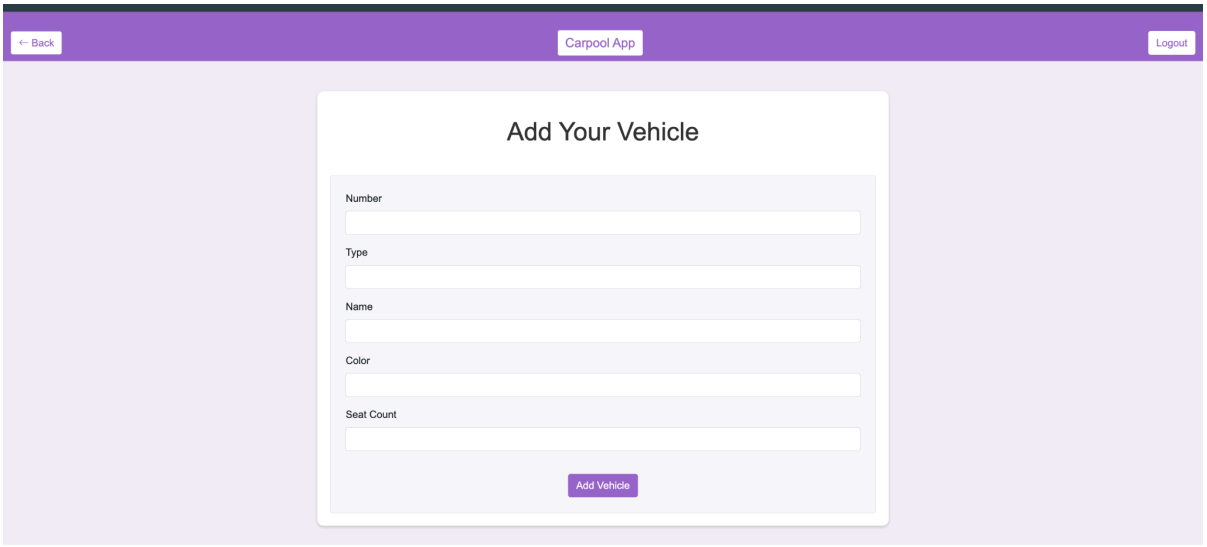


Fig19: Manage Vehicles[Add New Vehicle Details Form]

6.Create rides: [Setting pickup and Destination location on Map Using GoogleMapsAPI]

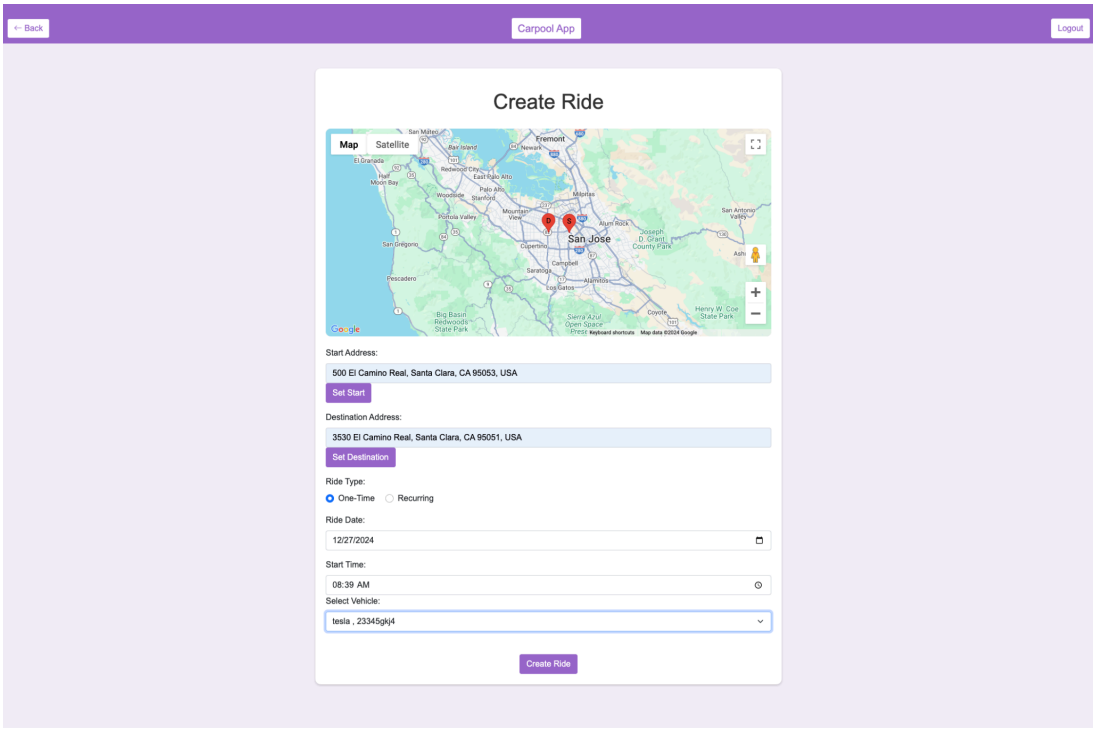


Fig20: Create Rides

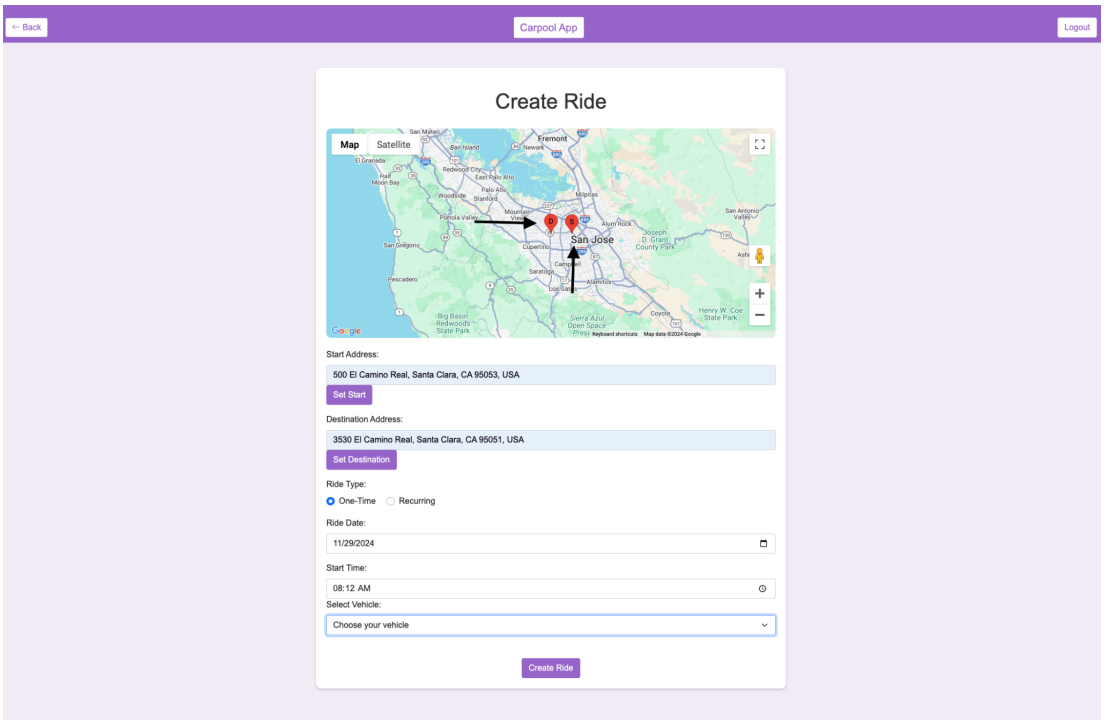


Fig21: Create Rides[selecting Locations]

- Selecting Rides for at least 1 month

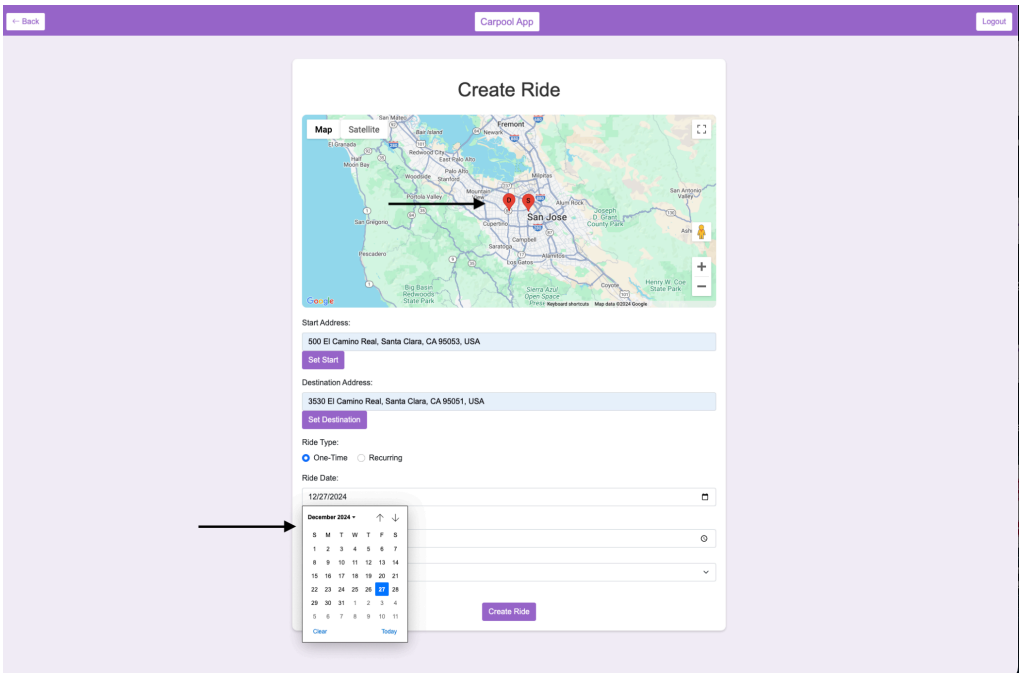


Fig22: Create Rides[selecting calendar for atleast 1 month in future]

- **For recurring rides:[Displaying Days of week]**

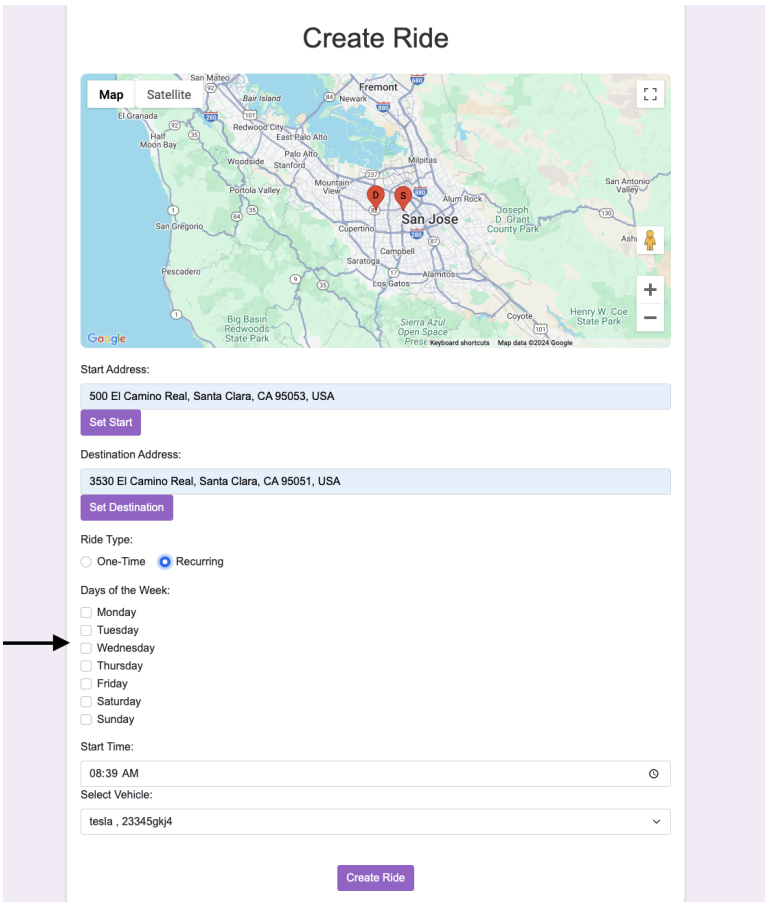


Fig23: Create Rides[Recurring rides creation]

- Rides created Successfully

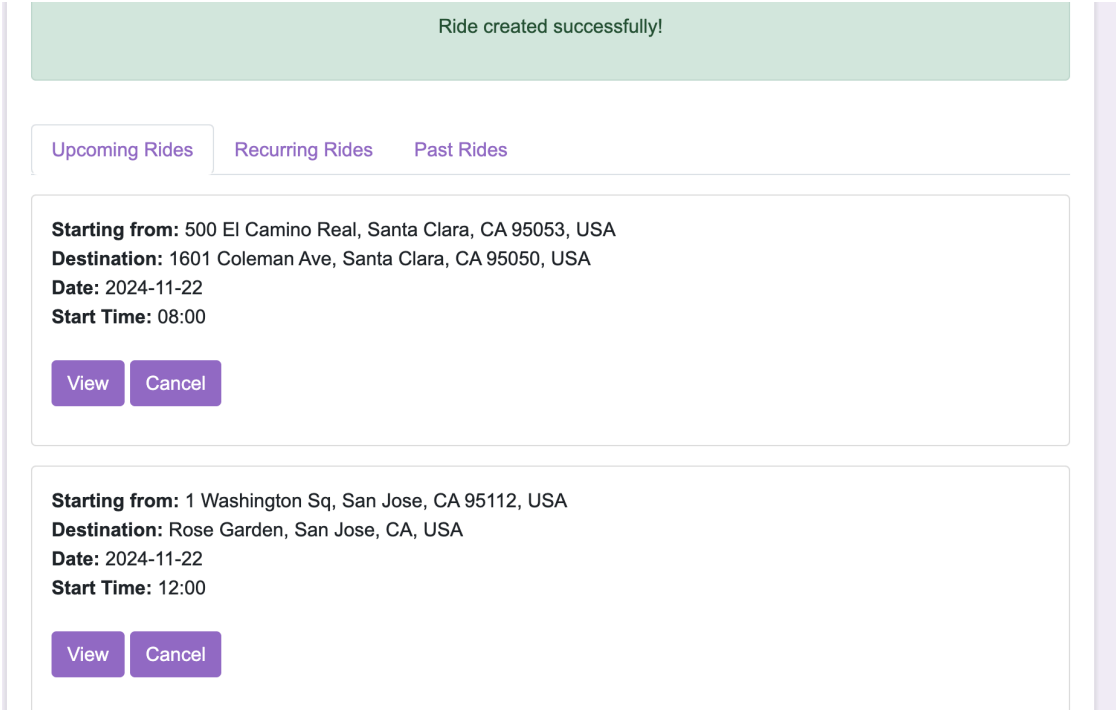


Fig24: Create Rides[Ride created Successful]

7. View Rides: [View Rides screen when there aren't any rides available]

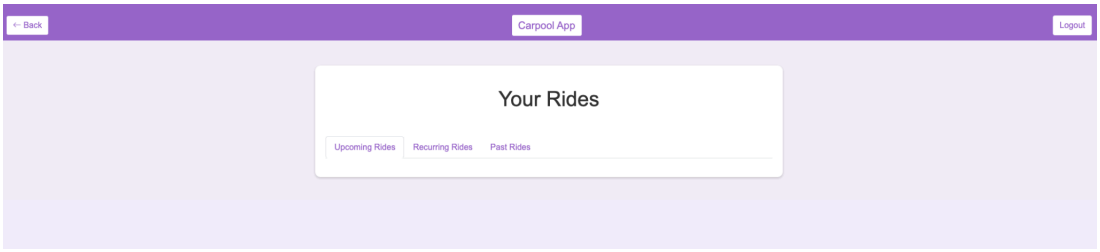


Fig25 : View Rides[Without any rides create/find]

- Same For Driver and Passenger if they haven't register or create ay rides

- **Upcoming rides:**

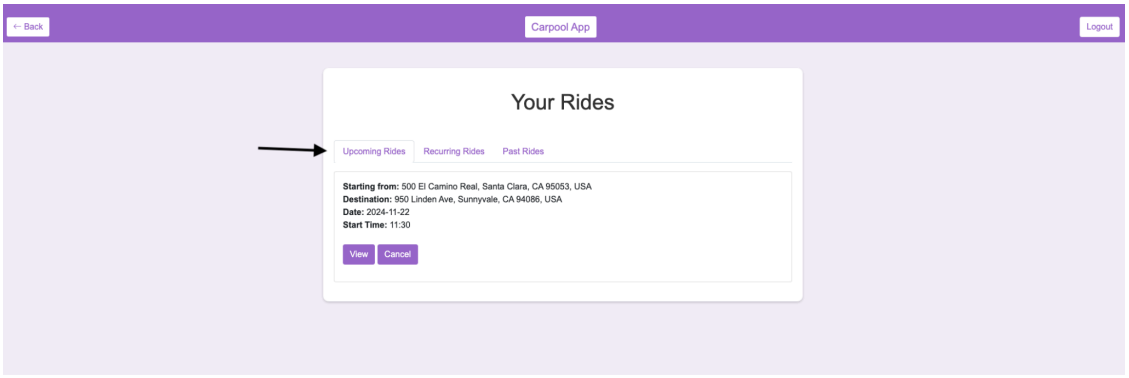


Fig26 : View Rides [Upcoming Rides for both Passenger and Driver]

- View Rides screen will have the same ride detail information for both Passenger and Driver.
- The Difference of the user can be seen when clicked on view.

- **Recurring Rides:**

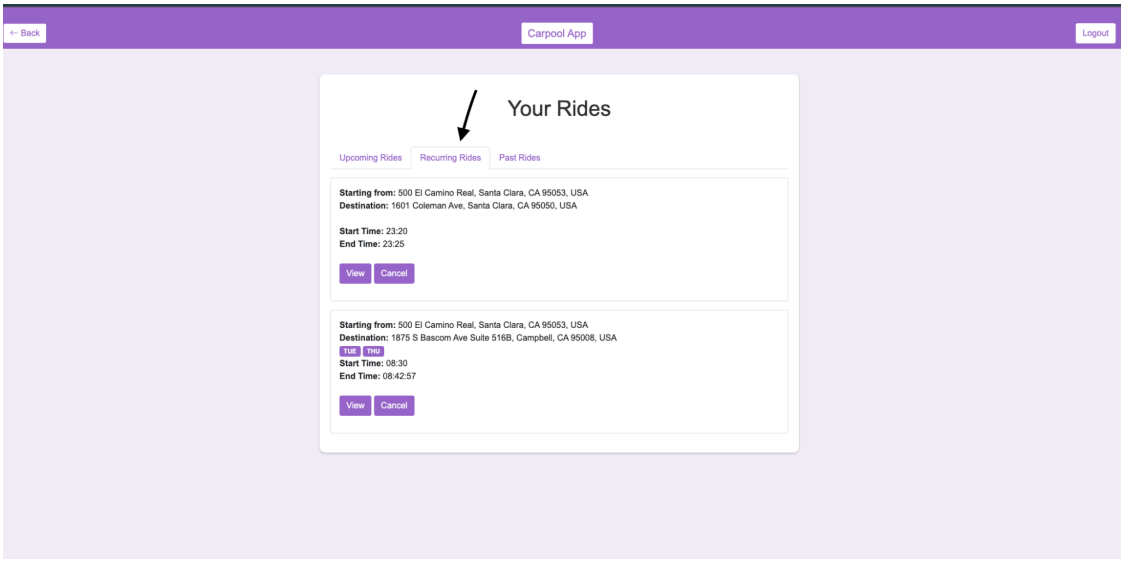


Fig27 : View Rides [Recurring Rides for both Passenger and Driver]

Past Rides:

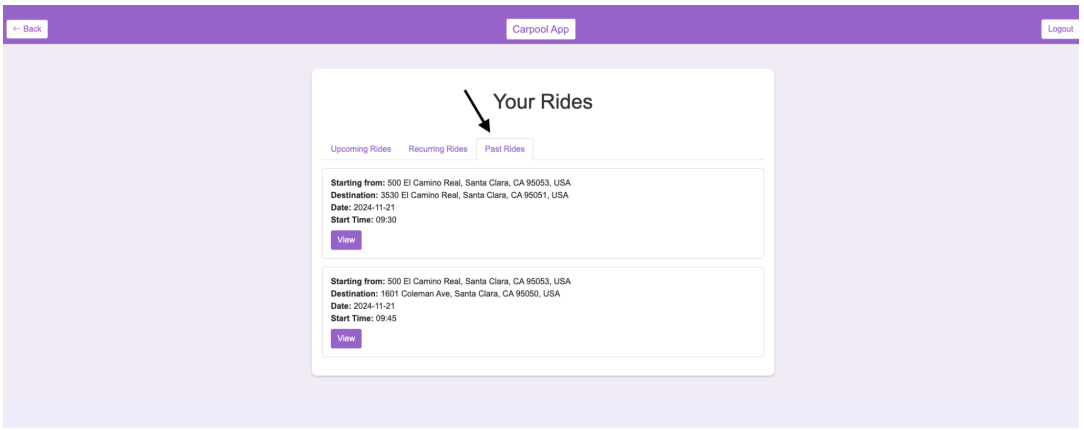


Fig28 : View Rides [Past Rides for both Passenger and Driver]

- **Ride Details:**

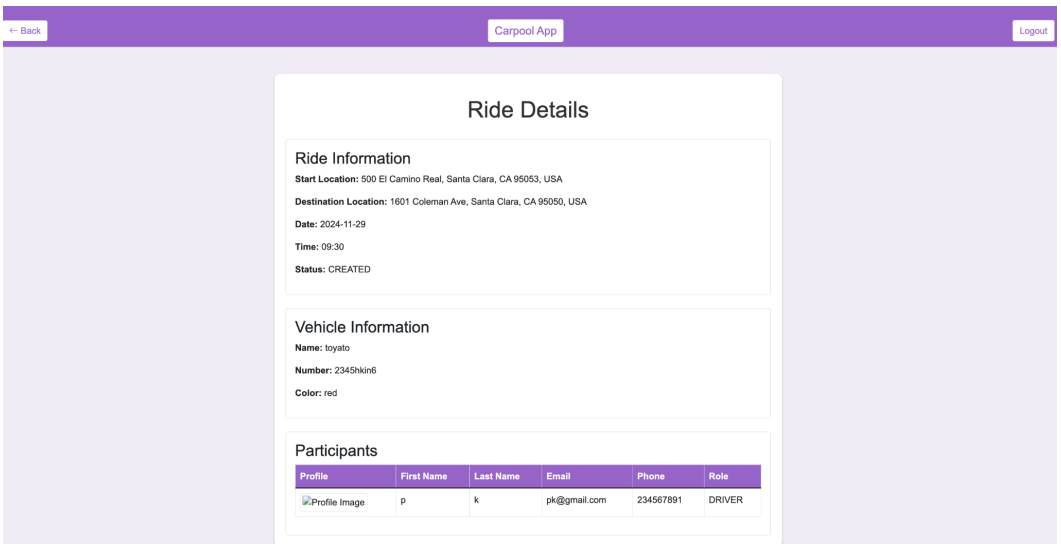


Fig29 : Ride Details [Ride Information for both Passenger and Driver]

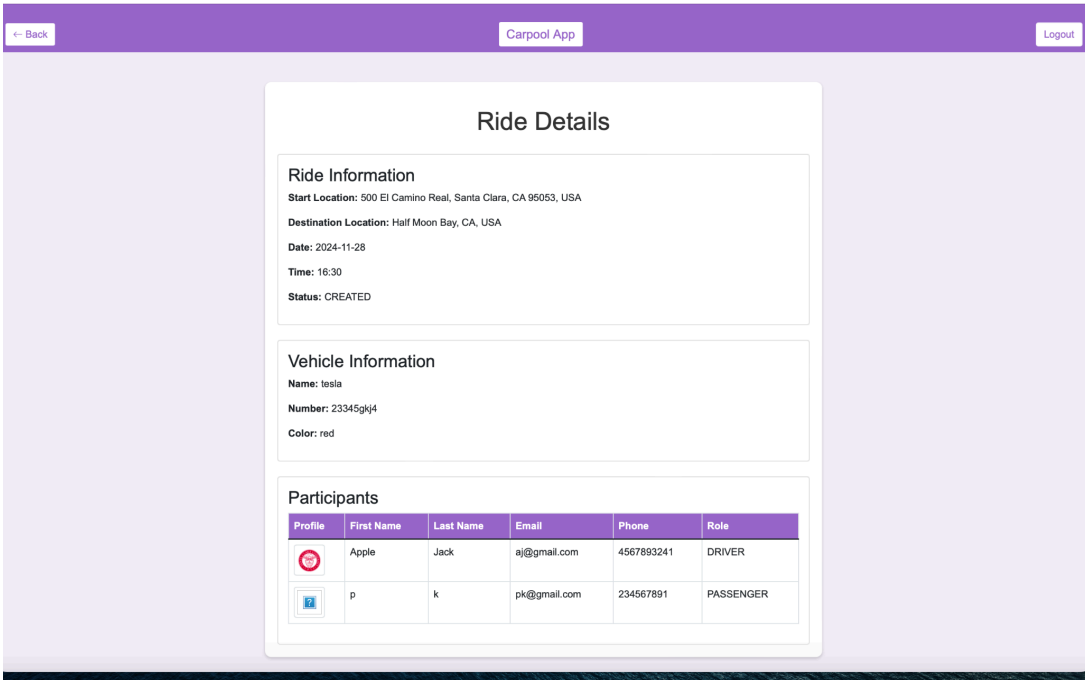


Fig30 : Ride Details [Ride Information for both Passenger and Driver when passenger added]

8.Find rides:[Match the rides for the particular distance with 15 mins end time]

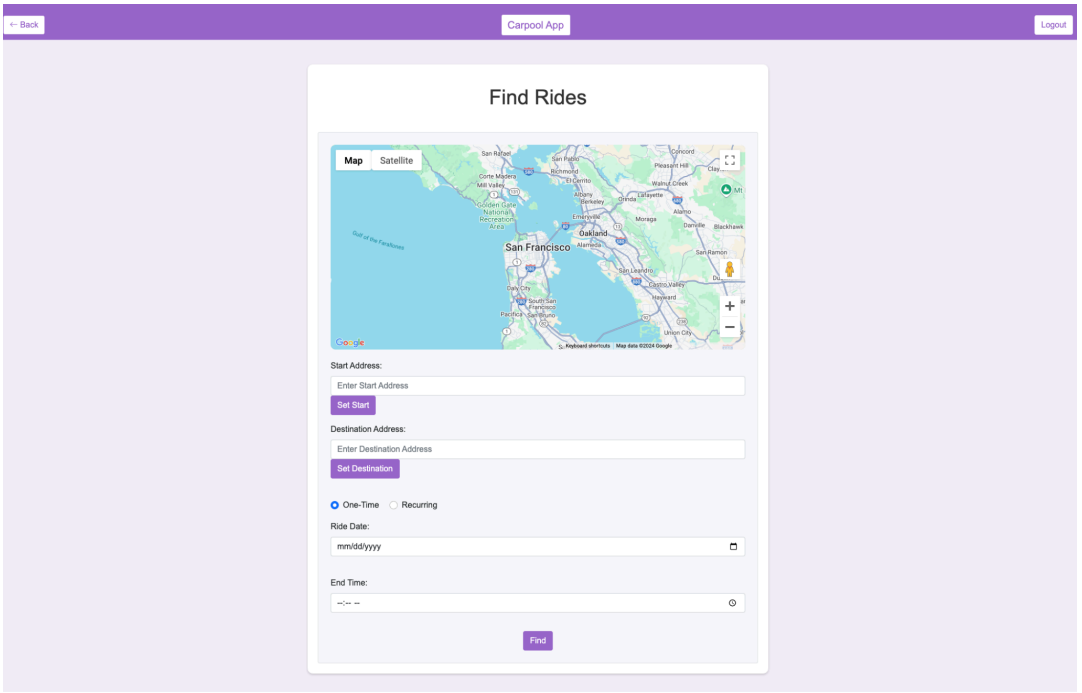


Fig31 : Find Ride[Without Any information]

- **Finding Rides for at least 1 month:**

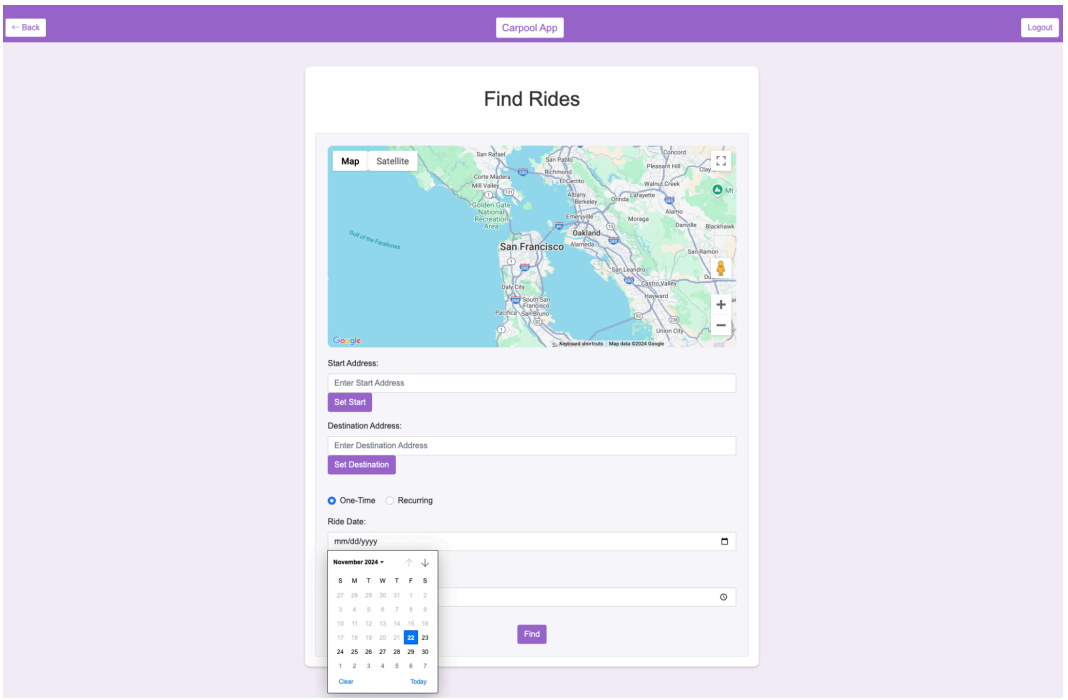


Fig32 : Calendar which blurred prev dates

- **Matching Rides Successful:**

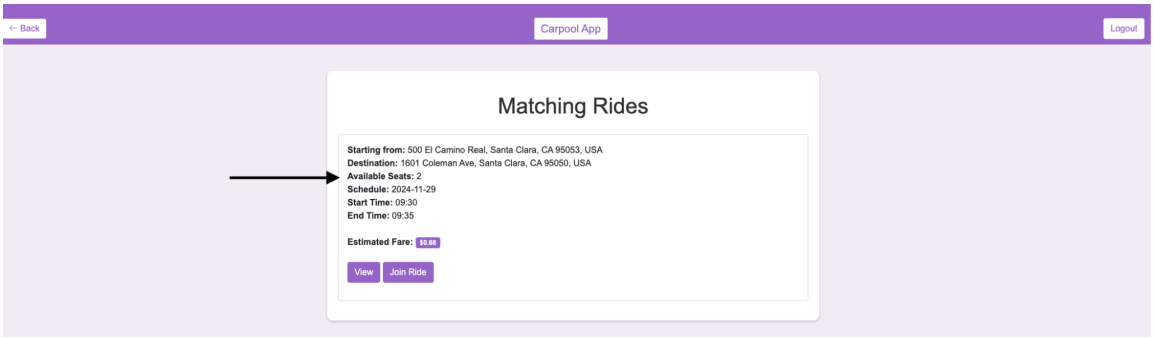


Fig33 :Matching Rides Successful showing ride detail with estimated fare

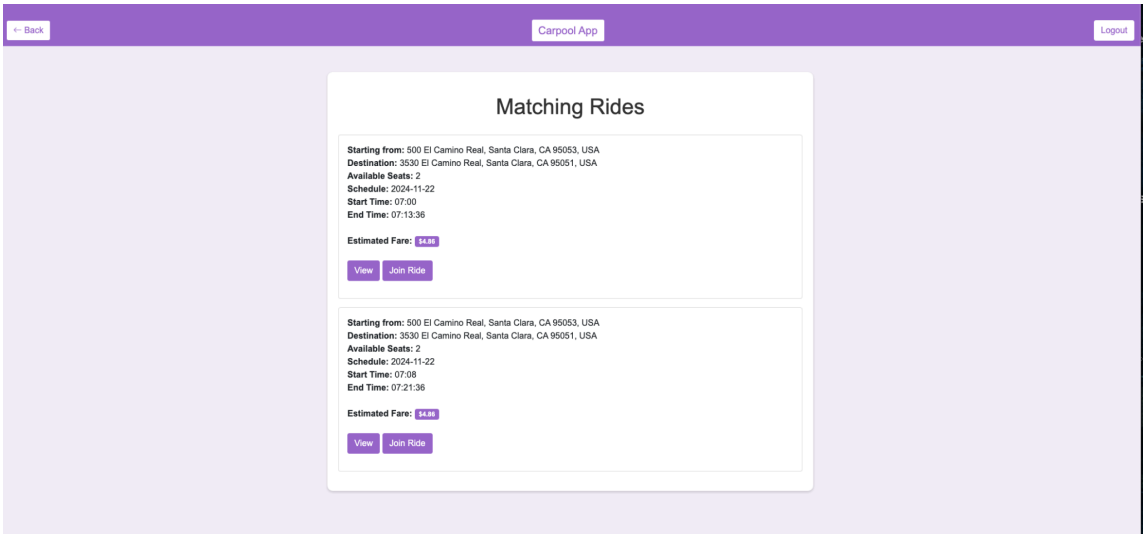


Fig34 : Showing Multiple Rides with estimated Fare to Join

- The above figures shows the matching rides which allow users to join in single/ multiple rides

- **Join Ride:**

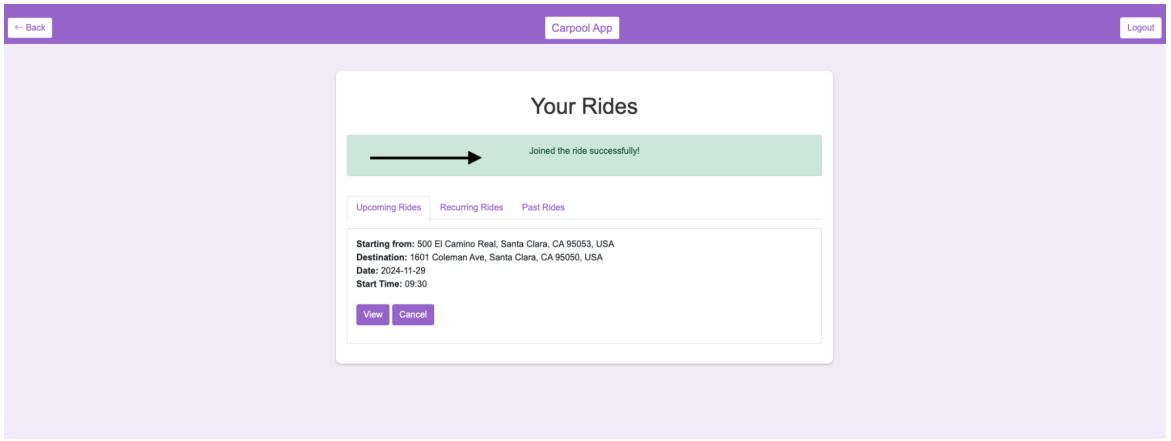


Fig35 : After Matching the Ride, User Joined Successfully

- **FindRide- View ride details are as follows:**

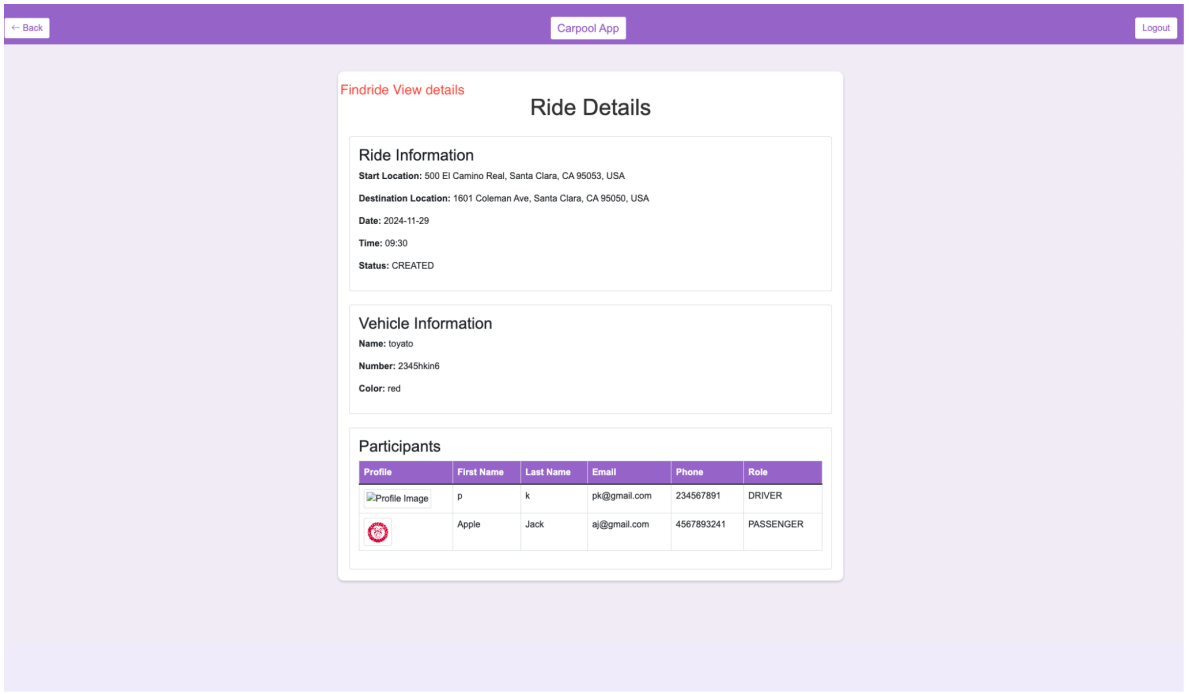


Fig36 : After Joining the Ride view details.

- The Ride Details will be displayed for both Driver/Passenger when there is an new user added to it

- **Ride Details Information:**

Passengers view:

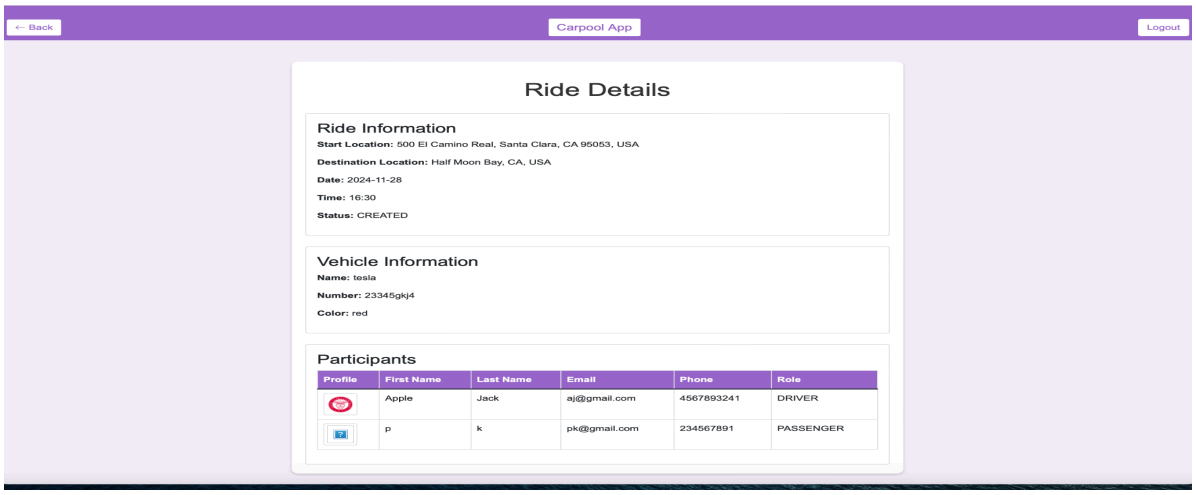


Fig37 : Passenger View Ride Details

- **If a passenger dropped from the ride. The ride still continues,passenger only be dropped**

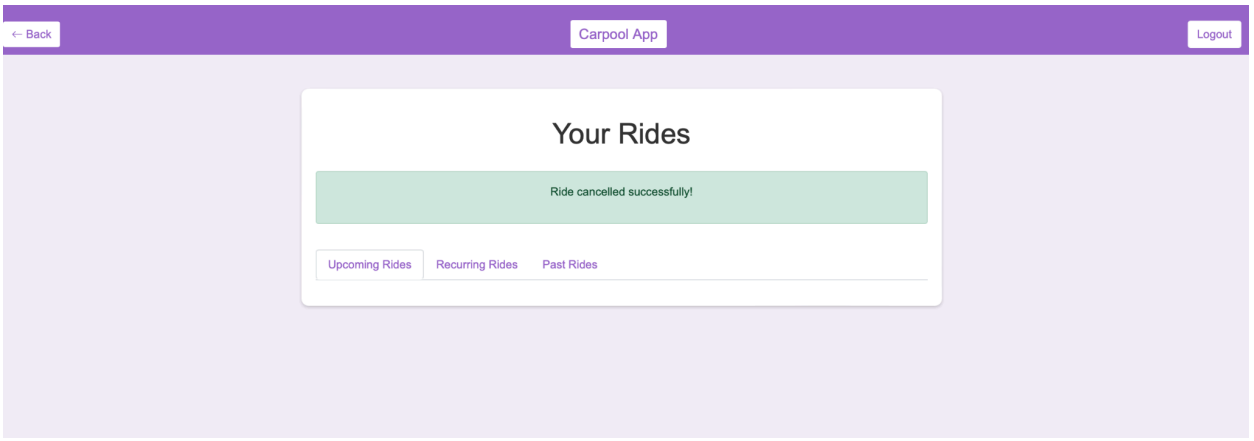


Fig38 : Ride Dropped by Passenger

- **If the driver cancels the ride, passengers are dropped from the ride. The ride will be dropped from Passenger**

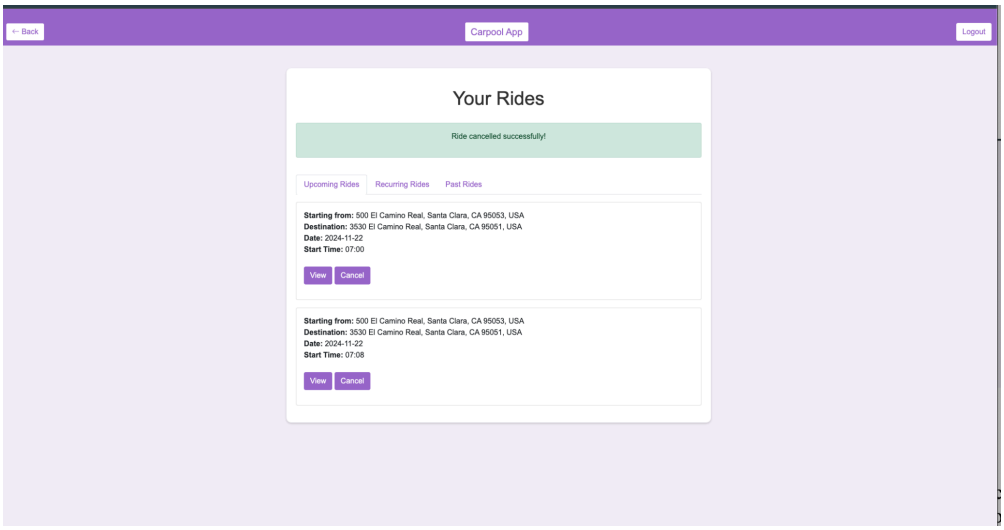


Fig39 : Ride Dropped by Driver

- **When a passenger drops from the ride**

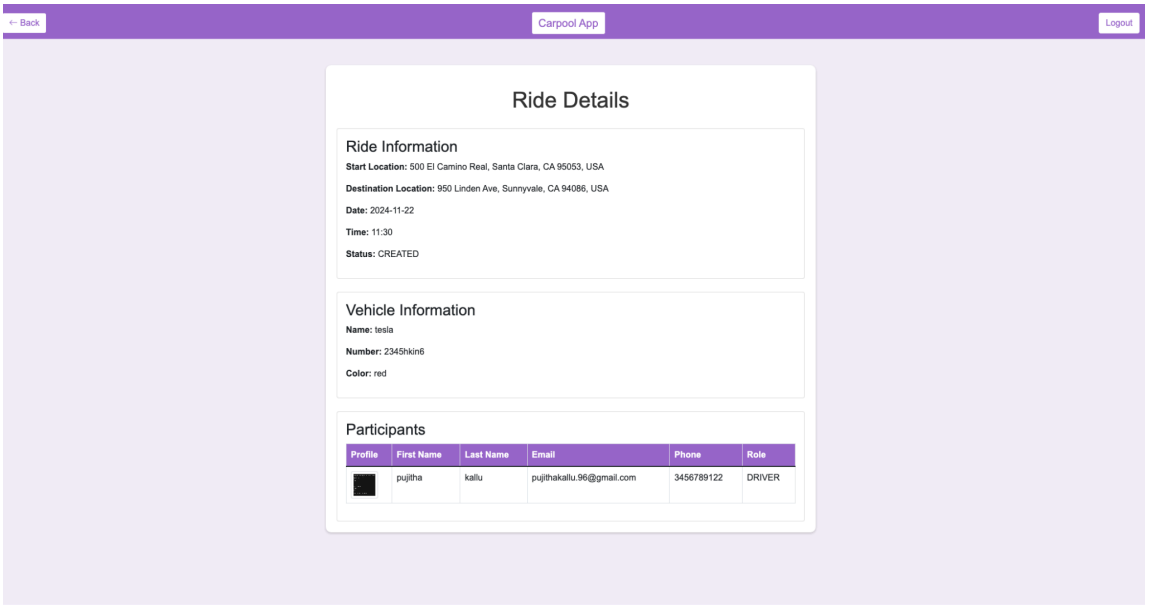


Fig40 : Driver View: Ride Dropped by Passneger

9. Wallets: [User Credit card Details to be updated]

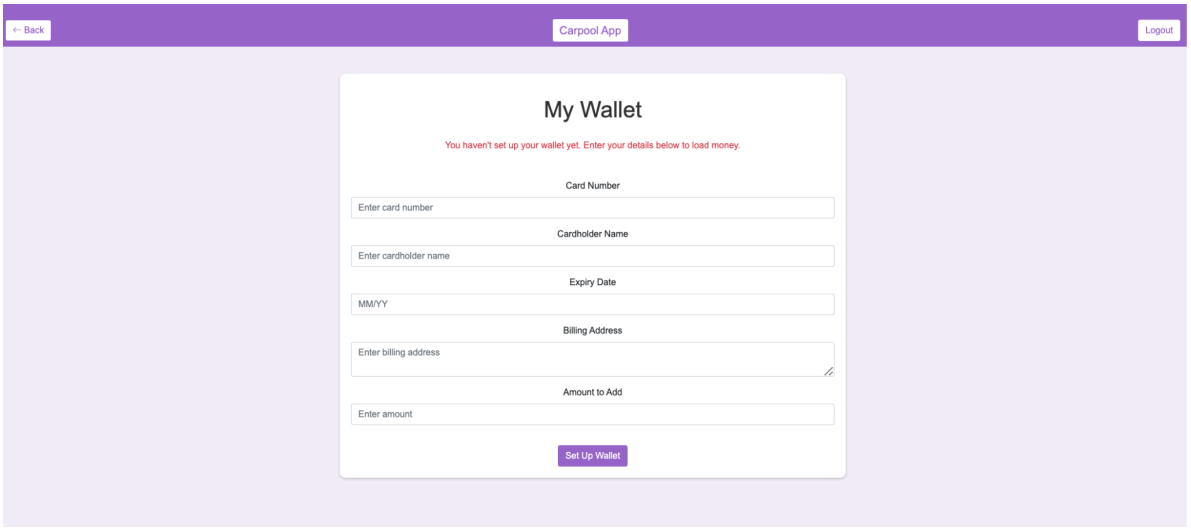


Fig41 : Adding New Credit card details Form

- **Update wallet details:**

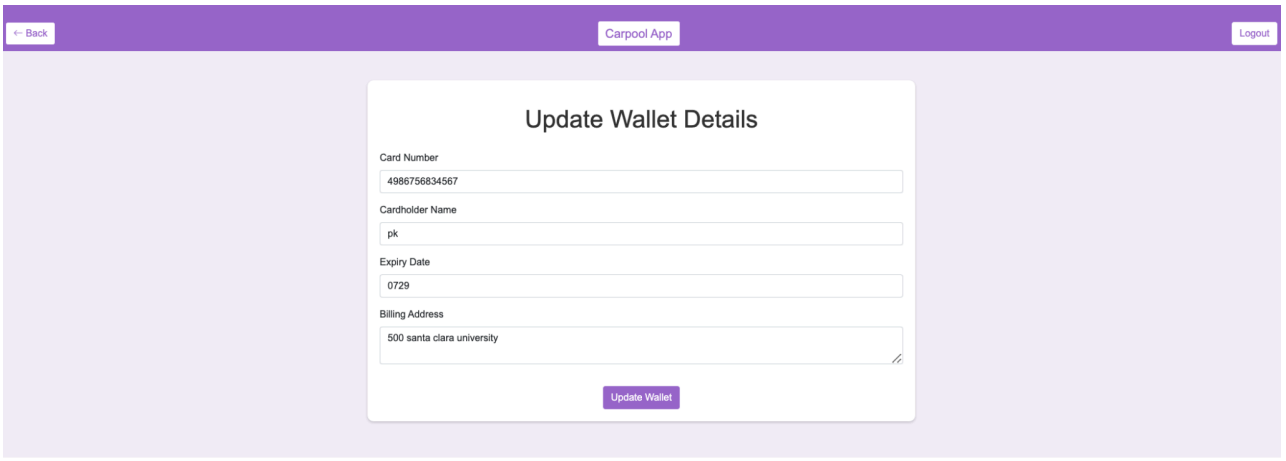


Fig42 : Updating Wallet Information

- **Added Amount to wallet**

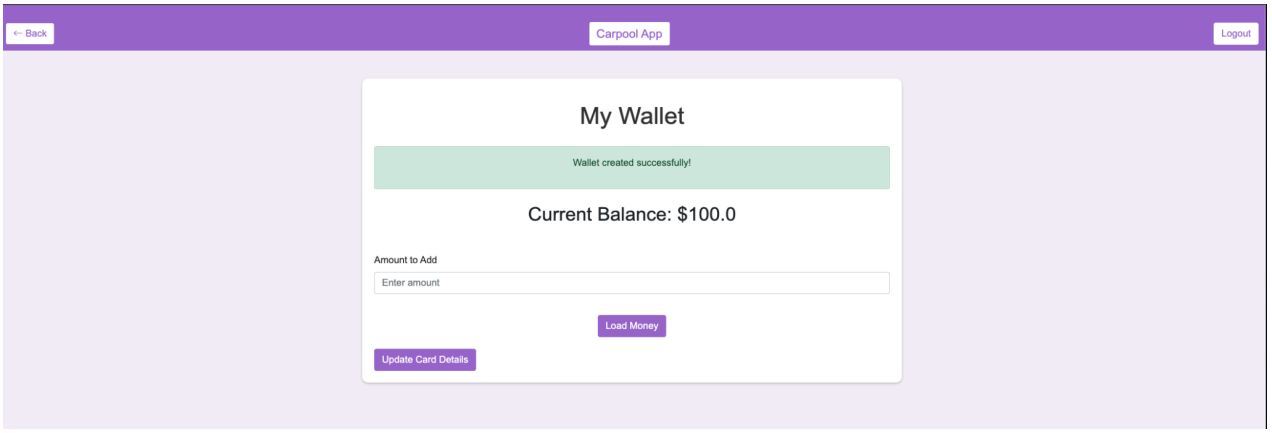


Fig43 : Loading Amount to My Wallet

- **Loadmoney to increase amount in the wallet**

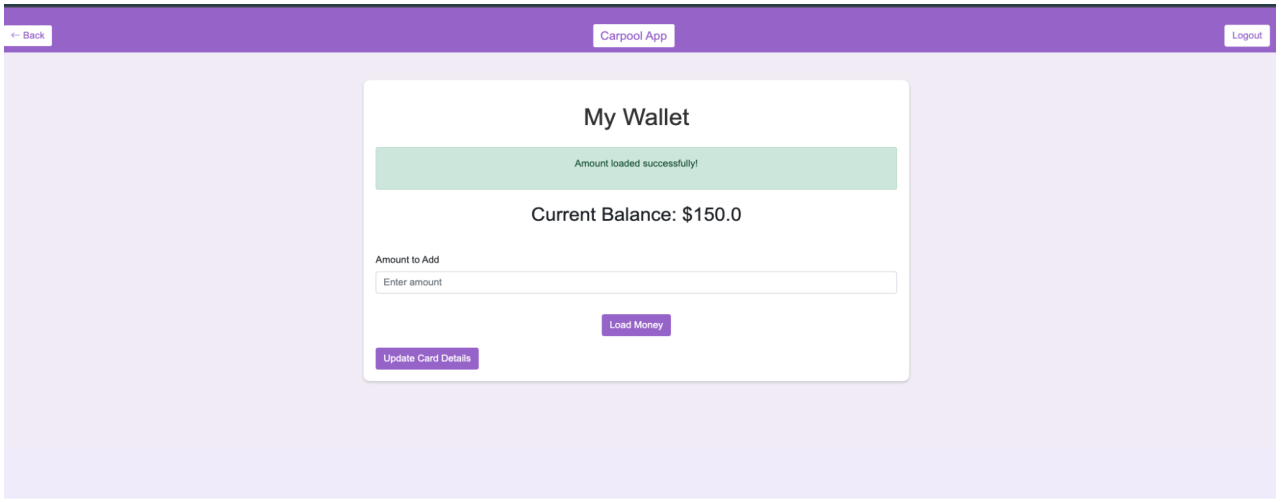


Fig44 : Adding extra amount to wallet

- **Balance After Paying owed amount**

For Driver:

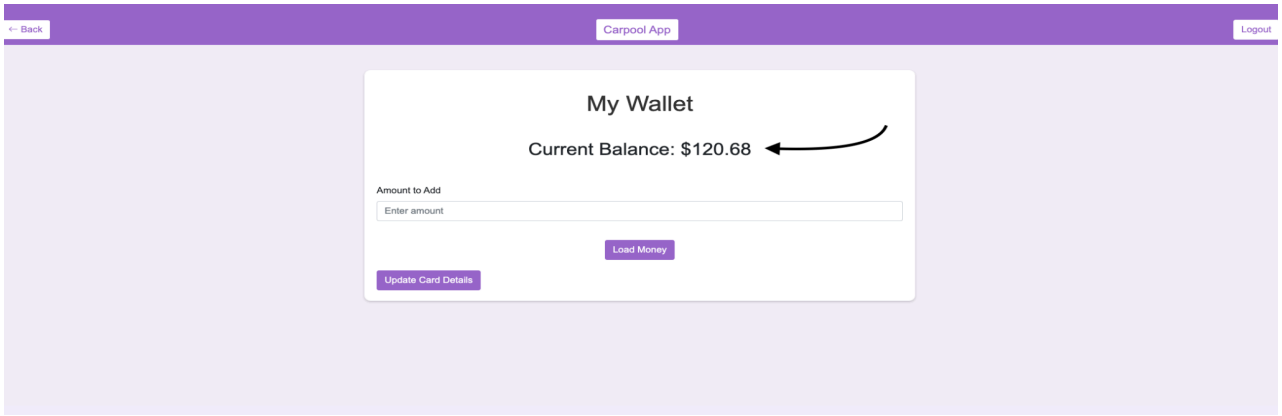


Fig45 : When Payment is done for past rides by passenger, Drivers Wallet

For Passenger:

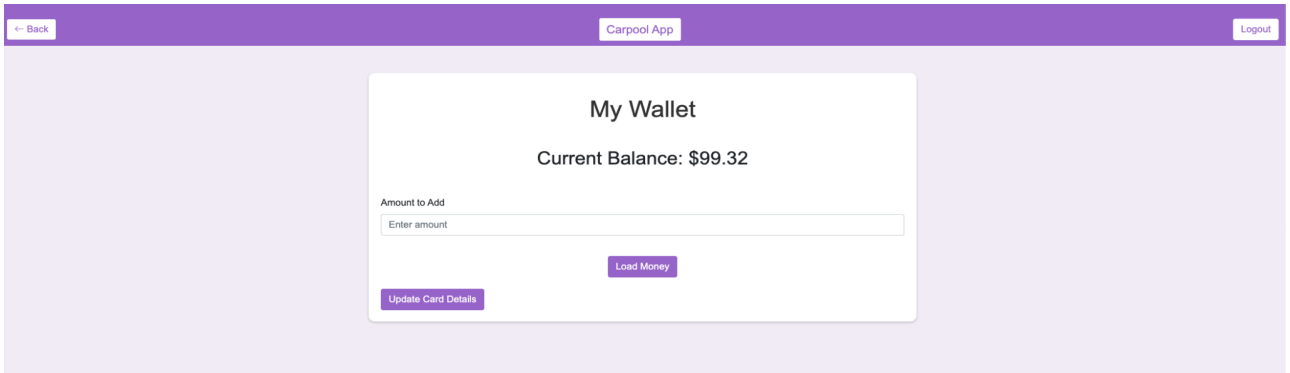


Fig46 : When Payment is done for past rides by passenger, Passenger Wallet

10.Payments: [Summary of Payments, owe,owed by passenger and driver]

- **Driver owed Amount:** [In green which mean driver gets paid]

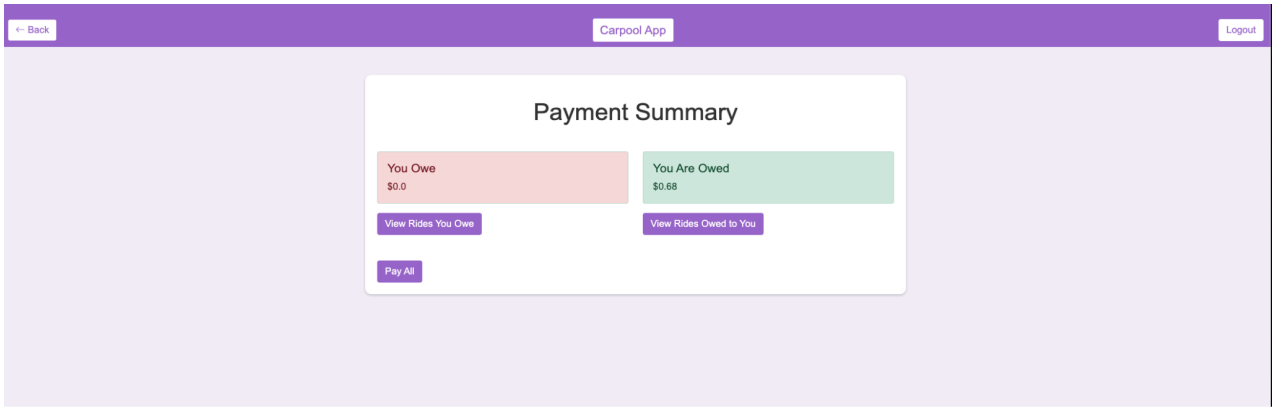


Fig47 : Payments screen when no ride occurred

- **Passenger owe Amount:** [In red which mean passenger has to pay]

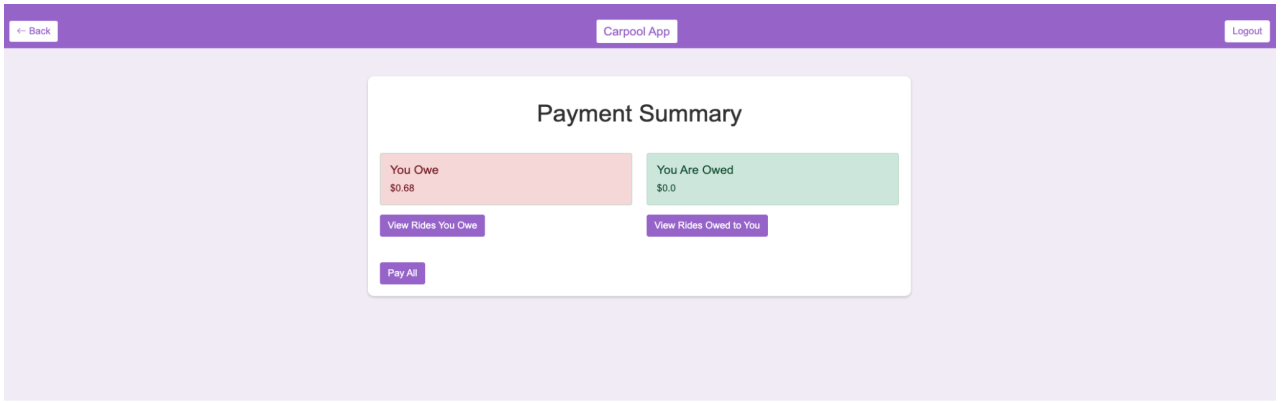


Fig48 : Payments screen when ride has completed for passengers

- **Redirecting to payment page from Payment to wallet page if card details are not available:**

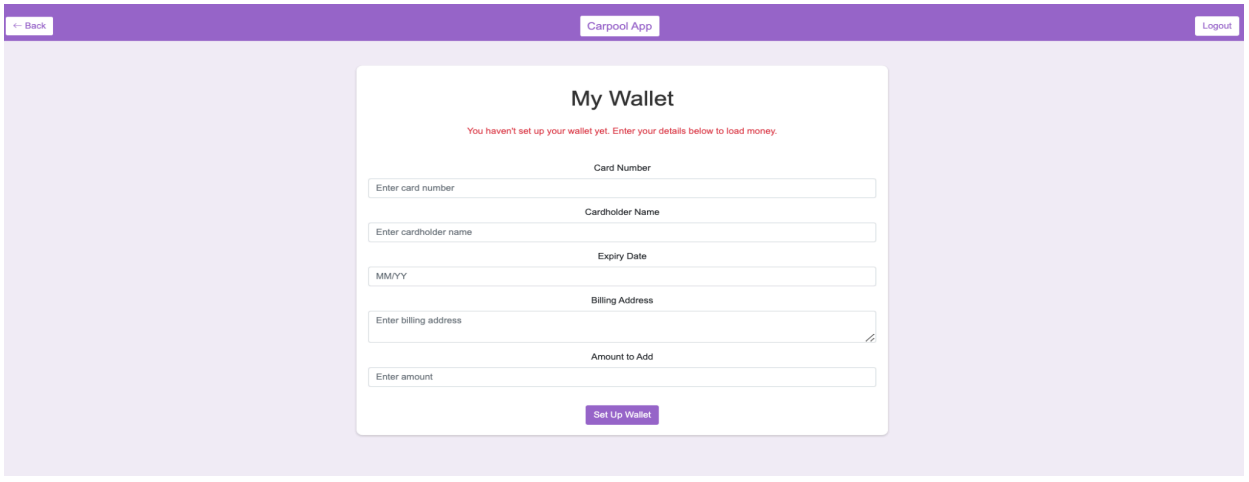


Fig49 : Redirect to Credit Card details when there is no card

- **Clearing all dues will reflect below screen:**

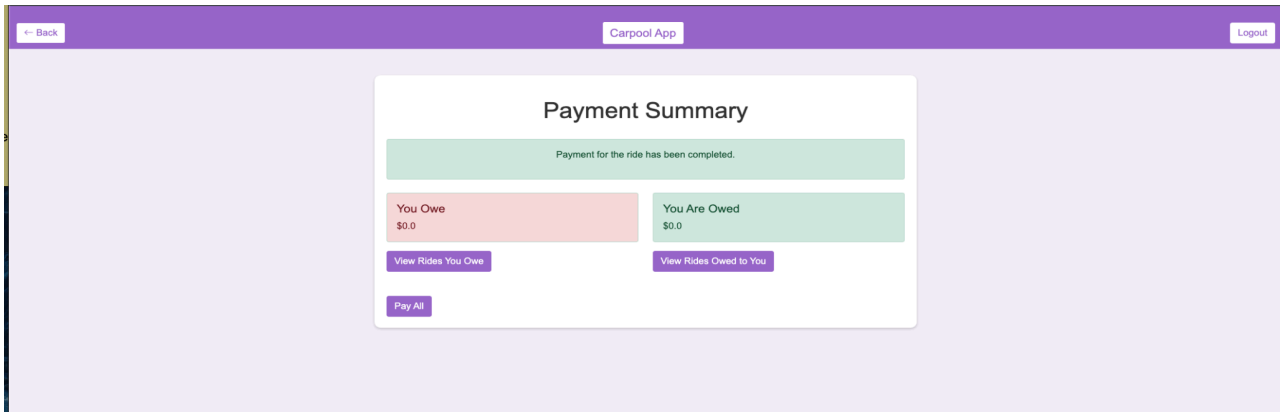


Fig49 : Payment gateway

- Once the payment has been done it will update on the driver's profile as well.

11.Logout:

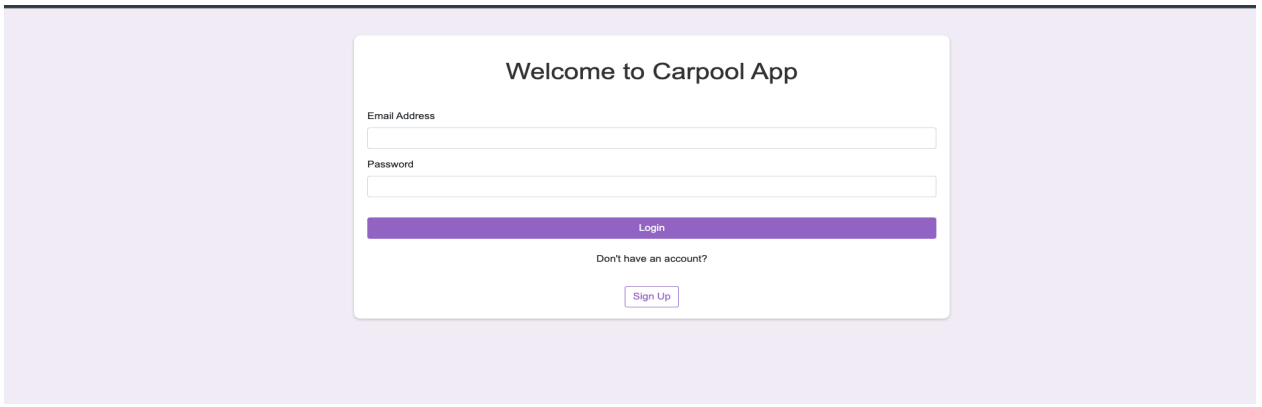
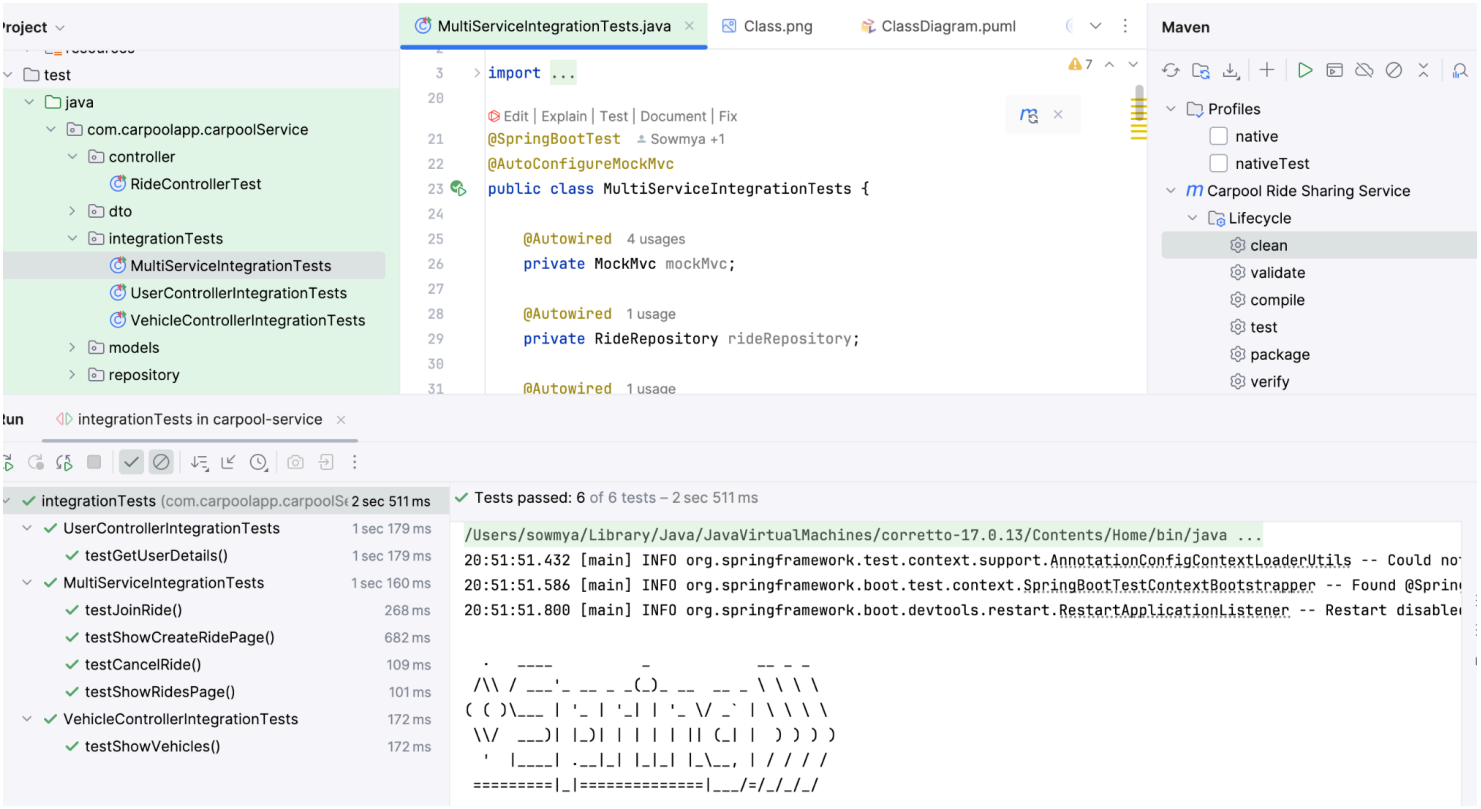


Fig50 : LogoutScreen

Integrating Testing:

Ran Test cases and verified if we passed all the test cases



Future Scope:

Future trends for carpooling applications include AI-powered ride matching, dynamic pricing, and route optimization, promoting sustainability with carbon footprint tracking and electric vehicle integration. Autonomous vehicles could enable driverless carpools, while blockchain enhances secure payments and transparency. Integration with public transit and last-mile mobility solutions offers seamless end-to-end travel. Cloud computing ensures scalability, and partnerships with smart cities and urban planning.

References:

1. **JavaSpringBoot :** https://spring.io/projects/spring-boot
2. **Java : Models,constructors :** https://www.w3schools.com/java/java_constructors.asp
3. **Mysql :** https://www.w3schools.com/MySQL/default.asp
4. **Maven :** https://maven.apache.org