A Project Report on

Cargo Together

By

Raj Deshpande

MCA – II, SEM – III

2024-25

To
Savitribai Phule Pune University, Pune

In Partial Fulfilment of the Degree of
Master in Computer Application (M. C. A.)

Under The Guidance Of
Prof. Poonam lalwani

Suryadatta Group of Institutes, Pune
Suryadatta Institute of Management and Mass Communication (SIMMC)

Date:-

<u>CERTIFICATE</u>

This is to certify that Mr. Raj Deshpande

has successfully completed his/her project work entitled **"** Cargo Together **"**

in partial fulfillment of MCA – I Semester-I program for the year A.Y. 2023-

24 .

He / She have worked under our guidance and direction.

Prof. Poonam Lalwani   **Dr. Manisha kumbhar**
(Project Guide)   **HOD-MCA,SGI**

Examiner 1   **Examiner 2**

Date :
Place :

Acknowledgment

Index

CHAPTER 1 : INTRODUCTION CARGOTOGETHER

CargoTogether is a mobile application designed to make transportation more accessible and convenient by connecting travelers and drivers for shared rides. **CargoTogether** aims to reduce travel costs, minimize environmental impact, and make long-distance commuting more efficient.

The app allows users to either offer or request rides, ensuring a more affordable and social travel experience. It also helps promote community-driven transport solutions by enabling carpooling for daily commutes, inter-city travel, or even casual road trips. **CargoTogether** focuses on safety, transparency, and convenience by incorporating features like driver verification, user ratings.

CargoTogether is a people transportation app designed to make travel more affordable, efficient, and eco-friendly by connecting drivers with passengers for shared rides. The platform enables carpooling for short or long-distance journeys, allowing users to find rides that match their routes and schedules. **CargoTogether** enhances the commuting experience by promoting a community-based transportation system, reducing costs, minimizing traffic congestion, and lowering carbon emissions.

CargoTogether is a modern people transportation app designed to connect drivers and passengers for convenient, cost-effective, and eco-friendly travel. Whether you're commuting to work, taking a road trip, or traveling between cities, **CargoTogether** makes it easy to find or offer rides with people heading in the same direction. By promoting shared rides, the app helps reduce transportation costs, minimize traffic congestion, and lower carbon emissions.

With its user-friendly interface and robust features, **CargoTogether** provides a seamless experience for both drivers and passengers, ensuring safety, transparency.

CargoTogether not only simplifies travel logistics but also fosters a sense of community by encouraging shared journeys and helping users make new connections along the way.

CargoTogether is designed as a peer-to-peer (P2P) transportation platform where individuals can connect to share rides, whether for daily commuting or long-distance travel. Similar to carpooling services, **CargoTogether's** mission is to reduce individual travel costs, minimize environmental impact, and make transportation more accessible.

Key Features :

- **User Profiles :** Allow users to create detailed profiles with travel preferences, reviews, and ratings.

- **Ride Sharing Matching :** A system to match riders and drivers based on travel routes, schedules, and preferences.

- **Publish Ride:** Enable drivers to publish their ride details, such as departure time, route, and available seats.

- **Multi-Language Support :** Make it accessible to a diverse user base by supporting multiple languages.

- **Booked Ride**: Allow users to book a ride easily through the platform.

- **Show Booked Ride History**: Display a history of rides booked by users for easy reference.

- **Ride Cancellation**: Provide options for both riders and drivers to cancel rides with appropriate policies.

Basic Workflow :

1. **User Registration :** Users sign up and create profiles. They can either register as riders or drivers.

2. **Ride Search & Booking:** Riders search for rides based on destination and timing. They book a ride, and drivers get notified.

3. **Booked Ride**: Allow users to book a ride easily through the platform.

4. **Show Booked Ride History**: Display a history of rides booked by users for easy reference.

5. **Ride Cancellation**: Provide options for both riders and drivers to cancel rides with appropriate policies.

1.1   Existing System

In the current system, traditional transportation methods include public transit, taxis, and private vehicle usage. However, these methods have limitations:

- **Public Transit :** Though relatively cost-effective, public transit can be unreliable, slow, and limited in terms of routes and schedules, making it less convenient for many users.
- **Taxis and Ride-hailing Services :** While more direct than public transport, taxis and ride-hailing services (like Uber or Lyft) are often expensive, especially for long-distance trips.
- **Private Vehicles :** Many people rely on private vehicles for commuting, but this results in under-utilized resources, higher costs for individual drivers, and increased traffic congestion.

Existing ride-sharing solutions, like **CargoTogether**, address some of these issues by offering ride-sharing services for long-distance travel.

Core Features of Existing Systems :

- **Scheduling and Route Planning:**
  Apps allow users to schedule rides in advance and plan the best route, often with options to optimize for speed, cost, or environmental factors.

- **Matching Algorithms:**
  Ride-sharing systems use algorithms to match drivers with riders based on trip details (origin, destination, timing).

- **Pricing Models:**

Apps either charge per kilometer/mile, or they use a pricing structure that includes fixed and variable costs depending on factors like distance, time of day, and demand.

- **Rating and Review Systems:**

Most ride-sharing services use rating and review systems for drivers and passengers to build trust and maintain quality standards.

- **Safety and Verification:**

For people transport, especially with strangers, safety is critical. Many apps verify users' identities through various methods, background checks, and provide safety features like location sharing.

- **Cancellation Policies:**

Services often have structured cancellation policies to protect both passengers and drivers and manage refunds or penalties as appropriate.

- **Environmental Sustainability:**

 Some platforms promote eco-friendly transportation, such as carpooling or carbon offset programs, to appeal to environmentally conscious users.

Current Limitations and Challenges :

- **Lack of Flexibility for Users:**

Most systems either focus on predefined routes (as with buses or shuttles) or demand-based routing (as with Uber). A system that offers more flexibility and customization could stand out.

- **Cost Management for Long-Distance Trips:**

While intercity options exist, long-distance or international ride-sharing remains underdeveloped due to cost, logistical challenges, and safety concerns.

- **Trust and Safety Concerns:**

Even with ratings and identity verification, people may be reluctant to ride with strangers due to personal safety concerns, especially on long trips.

- **Environmental Concerns and Efficiency:**

  Many people transport systems do not have optimized carpooling or shared ride systems that would maximize vehicle occupancy and reduce emissions.

Potential Unique Selling Points (USPs) for CargoTogether :

- **Flexible Long-Distance Ride-Share Focus:**

  If **CargoTogether** aims to go beyond intercity transportation and support regional or international travel, it could serve an untapped market.

- **Enhanced Safety and Trust Features:**

  Integrating additional safety features like real-time trip sharing, enhanced identity verification, and emergency contacts could increase user trust.

- **Dynamic Pricing Model:**

  Using a pricing model that adjusts based on distance, demand, and trip sharing could help attract more users by providing transparency and competitive rates.

- **Environmental Initiatives:**

   CargoTogether could incorporate features to offset carbon emissions or prioritize ride-sharing, setting it apart as an eco-friendly travel alternative.

- **Community Building Features:**

  To increase comfort with ride-sharing, **CargoTogether** could integrate community features, such as user groups based on common interests, to encourage more connected and comfortable travel experiences.

1.2   Need for the System

In the current transportation ecosystem, travelers primarily rely on traditional methods like public transportation, private vehicles, and taxi services for commuting and long-distance travel. While these systems are functional, they come with several limitations:

1. **Public Transportation:**
   - o **Inflexible Schedules :** Public buses and trains operate on fixed routes and schedules, which may not always align with users' specific travel needs.
   - o **Overcrowding :** In cities, public transit systems often become overcrowded, especially during peak hours, making travel uncomfortable.
   - o **Limited Reach :** Public transportation networks are often concentrated in urban areas, leaving suburban and rural regions with inadequate coverage.

2. **Taxis and Ride-Hailing Services :**
   - o **Costly for Long Distances :** While convenient, taxis and ride-hailing services like Uber or Lyft can be expensive, especially for long-distance travel.
   - o **Lack of Carpooling Options:** These services typically focus on individual rides rather than shared or carpooling options, leading to higher costs and more traffic congestion.

3. **Private Vehicles :**
   - o **High Ownership Costs :** Owning a car comes with significant costs, including fuel, insurance, maintenance, and parking.
   - o **Under-utilized Capacity :** Most vehicles are driven with empty seats, wasting fuel and resources.
   - o **Environmental Impact :** The use of private cars for solo trips contributes to traffic congestion and increased carbon emissions.

4. **Existing Ride-Sharing Solutions :**
    o Some platforms already exist to connect drivers with passengers for long-distance trips.
    o Many existing systems focus on intercity travel but do not effectively cater to short-distance, daily commuting needs.

CargoTogether addresses these limitations by offering a modern ride-sharing platform that connects drivers with passengers for both short and long-distance trips. It promotes efficient use of resources through carpooling, offers flexibility with real-time ride matching, and provides a more cost-effective and eco-friendly alternative to traditional systems.

1.3 Operating Environment Hardware and Software

Hardware and software Specification

1.3.1] Software Requirements

Technology: React Native

Client-Side Technologies: HTML, CSS, JavaScript , Ridux,
Async Storage,

Server-Side Technologies: MongoDB, NodeJS, ExpreshJS

Data Base Server: MongoDB

Operating System: Microsoft Windows

1.3.2] Hardware Requirements:

Processor: Intel Pentium 4 (or) Later.

Ram: 4GB Minimum, 8GB (recommended)

Hard Disk: 100 GB (or) Higher

Programming Languages:

- **Frontend** - HTML, CSS, JavaScript, Rieux, Async Storage

- **Backend** – MongoDB, NodeJS.

CHAPTER 2 : PROPOSED SYSTEM

Proposed System

The *CargoTogether* project, aimed at facilitating people transportation, could work as an innovative, shared transportation solution for connecting passengers to common destinations efficiently. Below is a detailed breakdown of the proposed system:

1. System Overview

- **CargoTogether** would serve as a digital platform that connects individuals who need transportation with drivers who have available seats, operating similarly to carpool or rideshare models.
- The focus is on both short-distance and long-distance travel, potentially catering to both daily commutes and intercity travel.

2. Key Components of the Proposed System

A. User Management

- **Passenger Profiles :** Passengers create accounts with basic information, travel preferences, payment methods, and ratings.
- **Driver Profiles :** Drivers register with details such as vehicle information, driving experience, insurance, and verification.
- **Verification Process :** Verification of driver identity, licensing, and vehicle safety to ensure safety and reliability for users.

B. Ride Matching Algorithm

- A robust ride-matching algorithm pairs passengers with drivers based on proximity, destination, timing, and travel preferences (like music choice, pet-friendliness).
- Dynamic pricing and route optimization are considered in the matching process to maximize efficiency and affordability.

C. Route Optimization

- The system uses real-time GPS data to optimize routes, reducing travel time, fuel costs, and emissions.

- Suggested pick-up/drop-off points could minimize detours, creating more convenient, streamlined routes for all riders.

D. Scheduling and Booking

- **Real-Time Booking :** Passengers can book available rides in real-time.

- **Advanced Booking :** Users can book rides in advance for specific dates and times, beneficial for planned travel like airport trips.

- **Recurring Rides** : For frequent trips (e.g., daily commuting), users can set up recurring rides with preferred drivers.

E. Cash Payment

- **Cost Sharing :** Passengers can share the cost of the journey, making the service affordable and eco-friendly.

F. Notifications and Communication

- **In-App Notifications :** Updates on booking status, ride confirmation, arrival times, and driver details are sent to both drivers and passengers.

- **In-App Messaging and Calls :** To ensure privacy and facilitate communication, users can contact each other through a masked number or chat feature.

G. Review and Rating System

- Post-trip ratings and feedback help maintain service quality. Both drivers and passengers rate each other, promoting accountability and trust.

H. Safety Features

- **Driver Verification and Background Checks :** A thorough driver verification process increases the reliability of the platform.

3. Technology Stack

- **Mobile App :** Built for iOS and Android with an intuitive, easy-to-use interface.

- **Backend Infrastructure :** Cloud-based for scalability, built using REST APIs, potentially on AWS or similar.

- **Database :** A secure database management system (e.g., PostgreSQL) to store user data and manage bookings.

- **Payment Processor :** Integration with a payment processor (like Cash only).

4. Business Model

- **Service Fee :** The platform charges a commission on each ride.

- **Subscription Model :** Premium subscriptions could offer additional features like priority booking.

- **Partnerships :** Collaborations with corporations for employee commuting solutions or with travel agencies for intercity rides.

5. Implementation Phases

- **Phase 1:** Core App Development (booking, user profiles, basic matching).

- **Phase 2:** Advanced Features (route optimization, dynamic pricing).

- **Phase 3:** Safety and Compliance Features.

- **Phase 4 :** Partnerships and Scaling.

6. Expected Benefits

- **For Passengers :** Cost-effective, flexible transportation with safety features.

- **For Drivers :** Additional income opportunities, flexible work schedule.

- Environmental Impact: Reduced carbon footprint through shared travel.

7. Challenges and Risk Mitigation

- **Safety Concerns :** Ensuring rigorous driver screening and secure communication.

- **Reliability of Services :** Implementing rating systems and trip guarantees for reliability.

- **Scalability :** Using cloud-based infrastructure to scale easily as demand grows.

2.2 Module specifications (Scope)

1. Objective :

- Develop a platform for arranging and coordinating people transportation. The focus could be on city-to-city or intra-city rides, optimizing cost, convenience, and ride availability.
- Offer a user-friendly experience, allowing users to find, book, and manage rides.

1. Module Specifications

Below are the modules that would be essential for the **CargoTogether** platform.

A. User Management Module

- **User Registration/Login:**
  - Standard registration via email, social media, or phone number.
  - Account verification with OTP or email.
  - Driver-specific registration with extra information, e.g., vehicle details and license verification.
- **User Profile Management:**
  - Allows users to update their information (profile picture, contact info, preferences).
  - Option for passengers and drivers to upload necessary documents for verification.
- **Driver Onboarding:**
  - Separate onboarding for drivers, including background checks, vehicle inspection, and document submission.

B. Search and Booking Module

- **Ride Search:**
  - Real-time search based on location, date, and time, with filters like "price," "ride type," and "reviews."
- **Ride Listings:**
  - For drivers to create ride listings specifying route, date/time, pricing, and available seats.

- **Booking System:**
  - Option to book directly or request approval from the driver.
  - Real-time updates for booking confirmation and status.

C. Scheduling and Routing Module

- **Route Optimization:**
  - Optimize routes based on demand, pickup/drop-off points, and traffic conditions.

- **Dynamic Scheduling:**
  - Allows drivers to update timings and locations if needed.

D. Cash Payment

- **Fare Estimation and Payment Processing:**
  - Estimate fares based on distance, time, and other variables.
  - Only hand over cash payments.

E. Ratings and Feedback Module

- **Review and Rating System:**
  - Post-ride ratings for both drivers and passengers.
  - Option for leaving feedback and reporting issues.

- **Issue Reporting and Resolution:**
  - Escalation channel for reporting and resolving disputes.
  - Automated and manual flagging of inappropriate behavior.

F. Notifications and Communication Module

- **In-App Messaging:**
  - Chat feature between drivers and passengers post-booking.

- **Push Notifications:**
  - Real-time updates on booking status, driver arrival, and trip completion.

- **SMS/Email Notifications:**
  - For confirmations, reminders, and emergency updates.

G. Admin Panel and Analytics Module

- **User Management Dashboard:**
  - Admin controls to monitor users, manage profiles, and handle complaints.

- **Ride Monitoring and Analytics:**

- View ride statistics, payment records, and user behavior insights.
- **System Reporting and Logs:**
  - Logs for security, payment history, and app performance.

2. Target Users:

- **Primary:** Individual passengers looking for affordable transportation options.
- **Secondary:** Drivers willing to offer rides to reduce costs or earn additional income.

3. Key Scope:

- **Core Features:** Enabling easy ride bookings, secure payments, and transparent communication between drivers and passengers.
- **User Interface:** Simple and intuitive interfaces on both web and mobile applications.
- **Reliability and Safety:** ID verification, driver reviews, ratings, and support to ensure a safe, reliable environment.

4. Platform Objective and Target Audience

- **Objective:**
  - Develop a people-transport service that connects drivers with passengers, allowing individuals to share rides for long or short distances. The platform should focus on affordability, accessibility, and user satisfaction.

- **Target Audience:**
  - **Primary Users:** Passengers looking for affordable travel options.
  - Secondary Users: Drivers interested in sharing empty seats to reduce travel costs or generate income.
  - **Tertiary Users:** Potential third-party partners, such as transport hubs, gas stations, or restaurants for collaborations and promotions.

5. Functional Scope

a) User Registration and Profile Management

- **User Registration:**
    - Enable registration via email, phone, and social media integrations.
    - Additional driver-specific registration requiring vehicle details, driver's license, and background checks.

- **Profile Management:**
    - Users can update personal details, set preferences, and manage privacy settings.
    - Driver profiles include vehicle details, availability status, and past ride records.

b) Ride Listing and Search

- **Ride Creation:**
    - Allow drivers to list their rides with details like route, departure time, number of available seats, price per seat, and amenities.

- **Ride Search and Filters:**
    - Passengers can search for rides based on criteria like pickup/drop-off location, date, price, and driver ratings.
    - Implement sorting and filtering options for personalized results.

c) Booking and Reservation System

- **Booking Process:**
    - A simple, user-friendly booking system allowing passengers to reserve seats on available rides.

- **Reservation Management:**
    - Real-time updates on booking confirmation, cancellations, and status changes.
    - Drivers can accept or reject passenger requests, with automated notifications to passengers.

d) Ratings, Reviews, and Support

- **Ratings and Reviews:**

- Implement a system for passengers and drivers to rate and review each other after rides, promoting transparency and accountability.

- **Customer Support:**
  - In-app support features for reporting issues, requesting refunds, or providing feedback.
  - An escalation system for addressing disputes and complaints.

6. Administrative and Operational Scope

a) Admin Dashboard and Monitoring

- **User and Ride Management:**
  - Admins can monitor users, manage bookings, and handle customer service inquiries.

- **System Monitoring:**
  - Real-time analytics for system health, user activity, and ride statistics.

- **Flagging and Compliance:**
  - Tools for detecting and flagging suspicious activity, enforcing compliance with community guidelines, and taking necessary actions.

b) Data Analytics and Insights

- **User Behavior Analysis:**
  - Collect and analyze data on user preferences, search trends, and booking habits to improve the platform and target services.

c) Marketing and Promotions

- **Marketing Campaigns:**
  - Tools for managing promotional campaigns, referral programs, and discounts to encourage user engagement.

- **Partnerships and Collaborations:**
  - Integration with local businesses and tourism services for partnerships, cross-promotions, and strategic collaborations.

7. Compliance and Security Scope

a) Data Security and Privacy

- **Data Encryption and Security Protocols:**
  - Ensure secure storage and transfer of personal data, payment information, and ride details.
- **Privacy Compliance:**
  - Adhere to regional data privacy regulations (e.g., GDPR, CCPA), allowing users to control their data and privacy settings.

b) Verification and Safety Standards

- **User Verification:**
  - Implement mandatory ID verification for drivers and optional verification for passengers.
- **Safety and Compliance Checks:**
  - Regular checks to ensure that drivers meet safety and service standards.
  - Enable a reporting feature for passengers to report misconduct or unsafe driving.

c) Liability and Insurance Policies

- **Insurance Integration:**
  - Offer insurance options for drivers and passengers, either through third-party insurance partners or in-house coverage.
- **Liability Management:**
  - Define the legal liability for the platform in case of accidents or disputes, clearly communicated to users through terms and conditions.

8. Technical Scope

- **Mobile and Web Applications:**
  - Develop dedicated mobile apps (iOS and Android) and a web application, ensuring a consistent user experience across platforms.
- **Backend Infrastructure:**
  - Cloud-hosted server environment to support high scalability, reliability, and data integrity.

9. Future Expansion Scope

- **Additional Features:**
  - Potential future features like carpooling recommendations, loyalty rewards, and premium memberships.
- **Regional and International Expansion:**
  - Scaling the service to cover more regions or even international routes, with necessary adjustments to regulatory requirements and language localization.

2.3 Objectives of System

The **CargoTogether** project, envisioned as a people transportation platform, aims to offer a streamlined and efficient way for users to find and share rides, reducing travel costs, improving connectivity, and promoting sustainable transportation. Here are the primary objectives of the system, broken down into detailed goals to ensure the platform is robust, user-centric, and meets diverse stakeholder needs.

1. Facilitate Affordable, Reliable, and Convenient Transportation

- **Cost-Effective Travel:** Enable users to find budget-friendly rides by allowing drivers to share available seats with passengers heading in the same direction. By distributing travel costs across multiple passengers, the platform makes transportation more affordable.

- Availability and Accessibility: Ensure users can access the platform via web and mobile, with easy search options to view and book rides anytime, optimizing convenience for both urban and rural users.

- **Service Reliability:** Establish a reputation for dependable service by ensuring ride availability, punctuality, and consistency in the quality of trips.

2. Enhance User Safety and Trust

- **Identity Verification:** Require thorough verification for both drivers and passengers, including identification checks, vehicle verification, and driver license validation to maintain a secure environment.

- **Ratings and Feedback:** Allow users to rate and review rides, fostering a community of trust and ensuring drivers and passengers adhere to expected standards.

- **In-App Safety Features:** Include emergency features such as an SOS button, ride-sharing with contacts, and reporting options to address safety concerns proactively.

3. Offer a Seamless, User-Friendly Experience

- **Intuitive Interface:** Design a clean, intuitive interface that enables users to search, book, and manage their rides with ease. This includes user-friendly navigation for drivers creating ride listings and for passengers seeking available rides.
- **Efficient Booking System:** Enable quick booking, rebooking, and flexible cancellation processes that allow users to plan and adjust trips with minimal hassle.
- **Personalized Recommendations:** Implement algorithms that suggest rides based on user preferences, recent trips, and travel history, making the booking process faster and more relevant.

4. Promote Sustainability and Environmentally Friendly Travel

- **Reduced Carbon Footprint:** Encourage shared rides to reduce the number of vehicles on the road, helping to lower emissions and contributing to greener transportation.
- **Incentives for Eco-Friendly Choices:** Offer rewards or discounts for users choosing to carpool regularly, incentivizing a shift toward more sustainable commuting habits.

5. Provide Scalable and Flexible Scheduling for Drivers and Passengers

- **Flexible Ride Listings:** Allow drivers to create and update ride listings with options for scheduled, recurring, or last-minute trips to accommodate a wide range of transportation needs.

- **Dynamic Routing and Pickup Options:** Enable route flexibility, including multiple pick-up and drop-off points, so drivers can optimize their route based on passenger demand and traffic conditions.
- **Real-Time Updates and Notifications:** Provide timely notifications for ride confirmations, schedule changes, and arrival reminders to ensure users are informed and prepared.

6. Support Robust Admin and Management Capabilities

- **Comprehensive Admin Dashboard:** Develop an admin panel for monitoring user activities, handling disputes, managing drivers, and overseeing ride logistics.
- **Automated Compliance and Fraud Detection:** Employ AI-driven systems to detect and prevent fraudulent behavior, maintaining the integrity and compliance of the platform.

7. Enable Strong Community Engagement and Support

- **Community Building Features:** Facilitate a sense of community among users with social features such as public profiles, user status updates, and the ability to mark drivers/passengers as "favorites" for future rides.
- **In-App Support and Help Desk:** Provide a dedicated support system within the app, offering FAQ sections, chatbots, and customer service for resolving queries and disputes.
- **Feedback Loops for Continuous Improvement:** Encourage user feedback and incorporate it into regular platform updates to ensure the system evolves with user needs.

8. Ensure Compliance with Legal and Regulatory Standards

- **Regulatory Compliance:** Adhere to local laws regarding transportation, data privacy, and passenger safety, ensuring the platform meets regional regulatory standards.
- **Insurance and Liability Management:** Partner with insurance providers to offer coverage options for both drivers and passengers, protecting all parties in case of accidents or incidents.
- **Data Protection and Privacy:** Follow data protection laws, ensuring user data privacy and security in all interactions.

---

9. Facilitate Expansion and Scalability for Future Growth

- **Scalable Infrastructure:** Build the platform with scalable cloud infrastructure to support an increasing user base as the platform grows regionally and potentially internationally.
- **Multi-Language and Multi-Currency Support:** Design the platform to support multiple languages and currencies for expansion into new markets.
- **Partnership Integrations:** Enable partnerships with businesses like gas stations, eateries, or local transit for enhanced user experience and potential growth avenues.
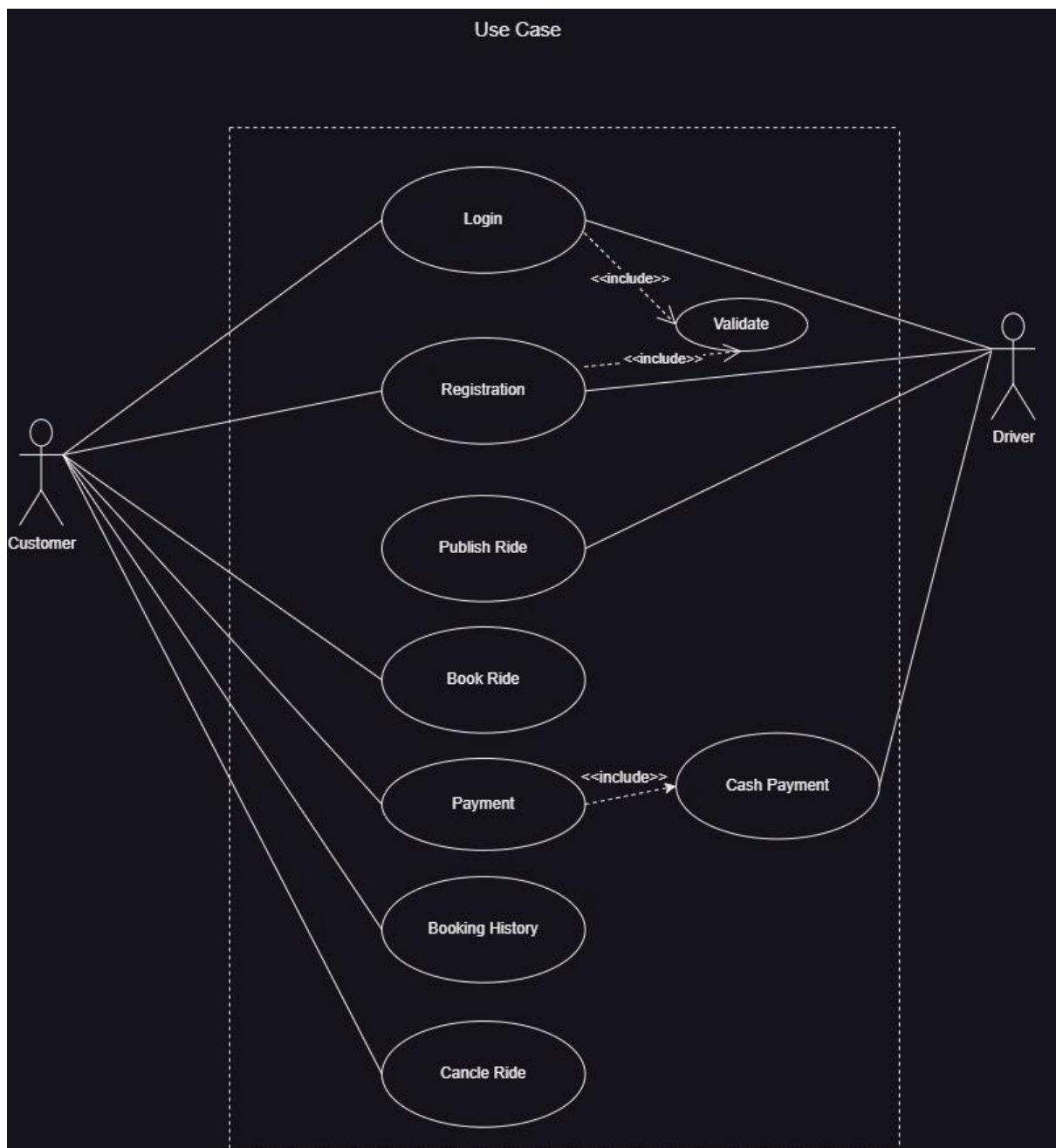
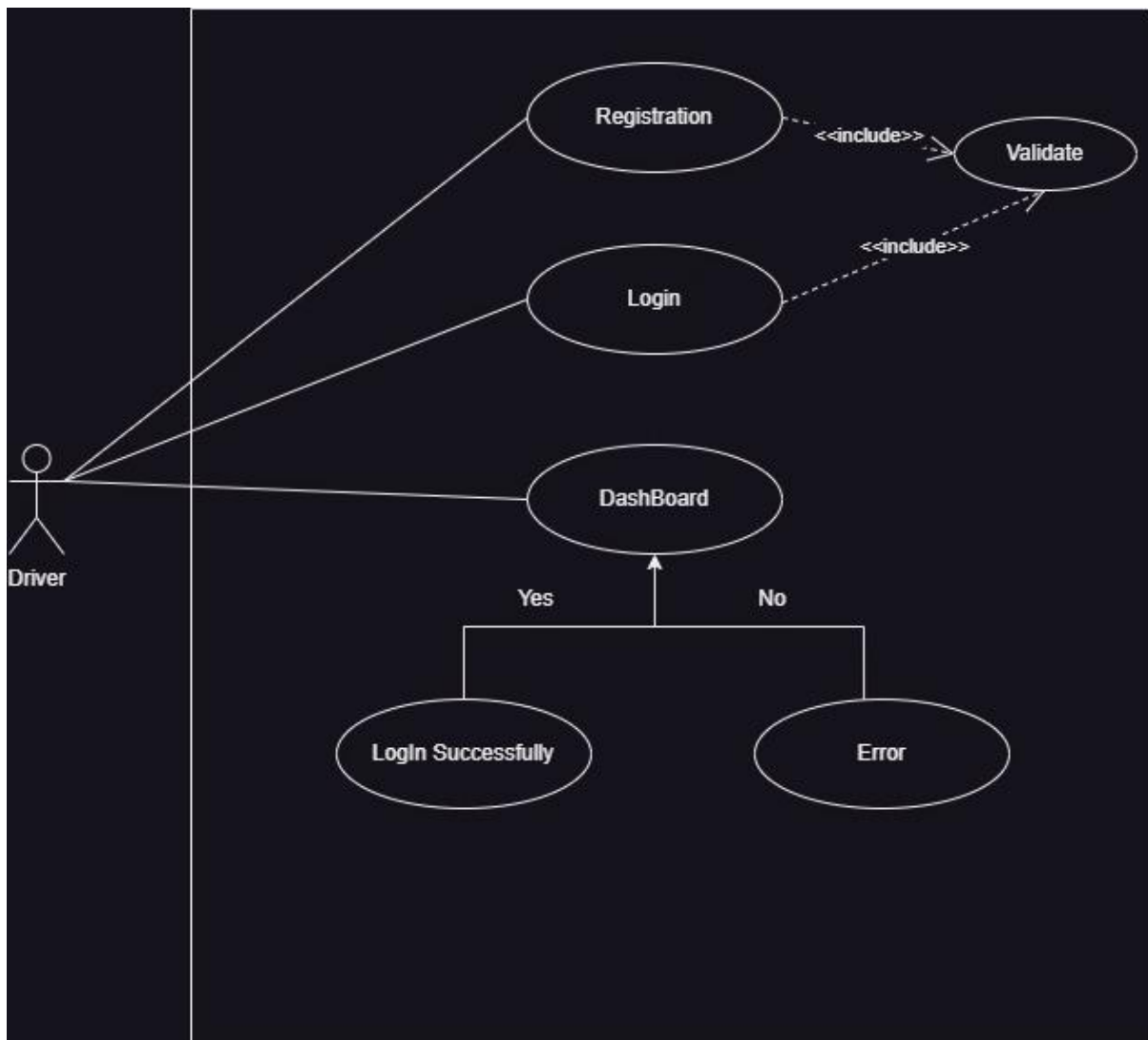# CHAPTER 3 : ANALYSIS & DESIGN

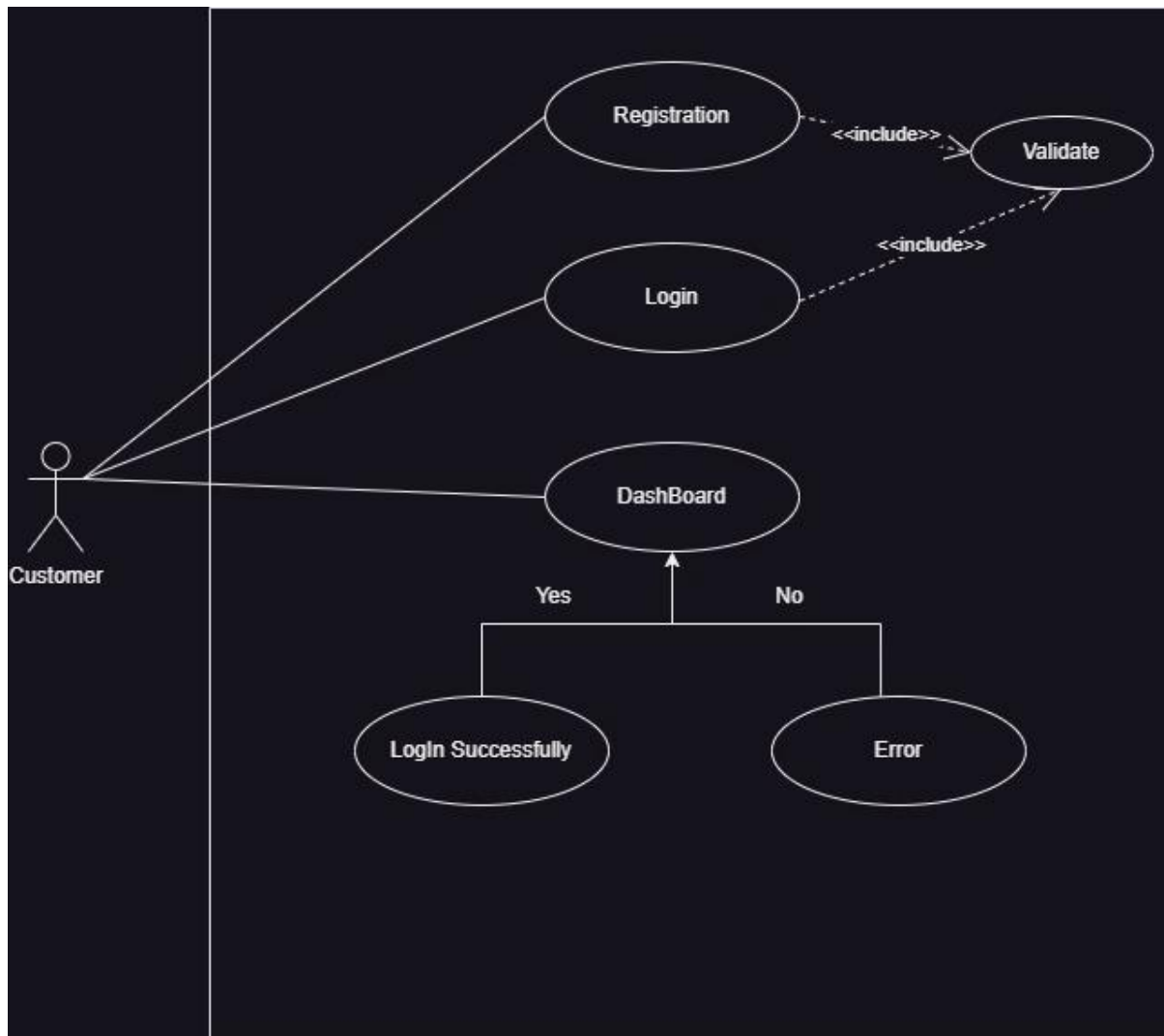## 3.1 Use Case Diagrams

A. Main Use Case Diagram
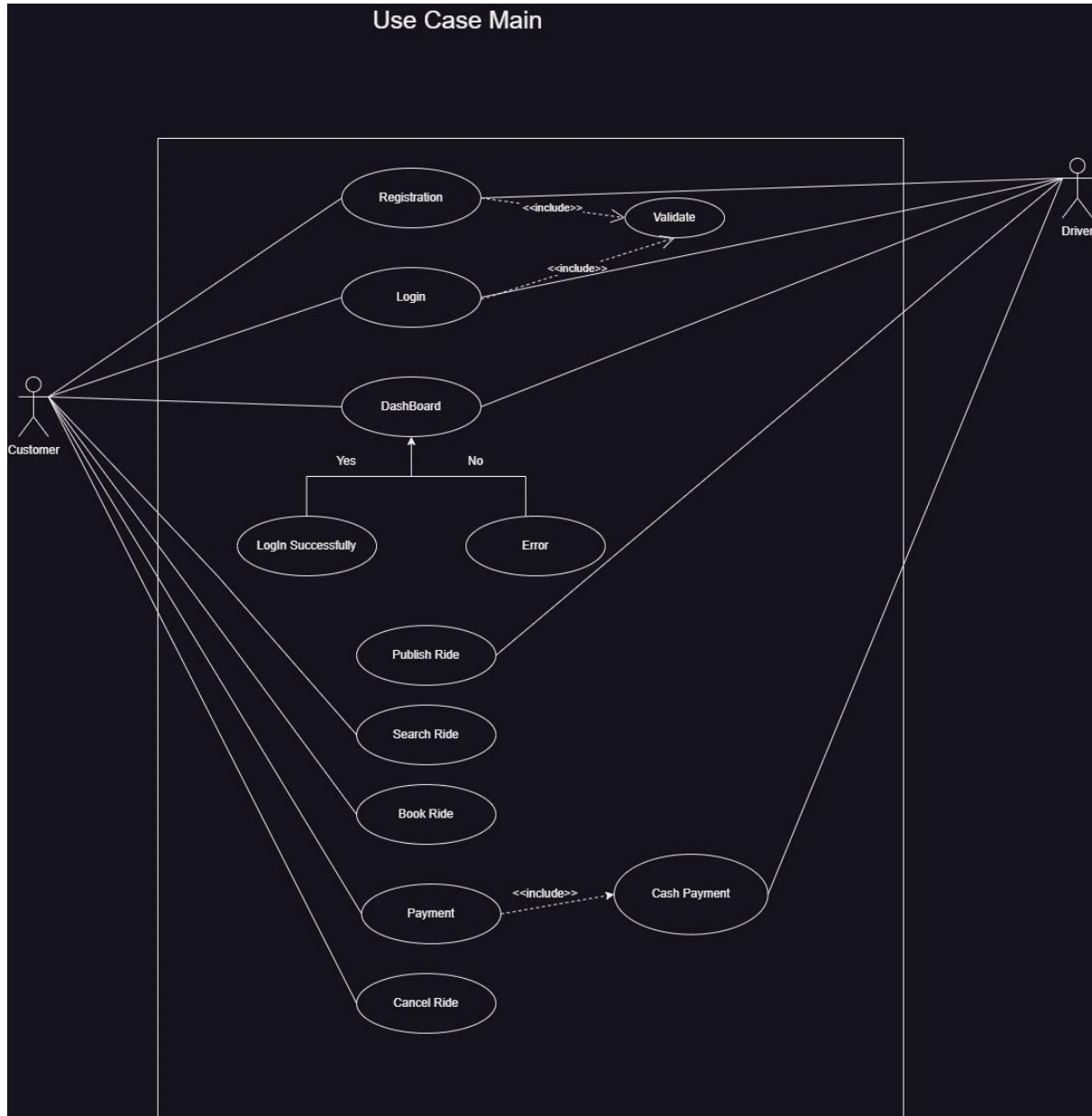
B. USE Case Diagram
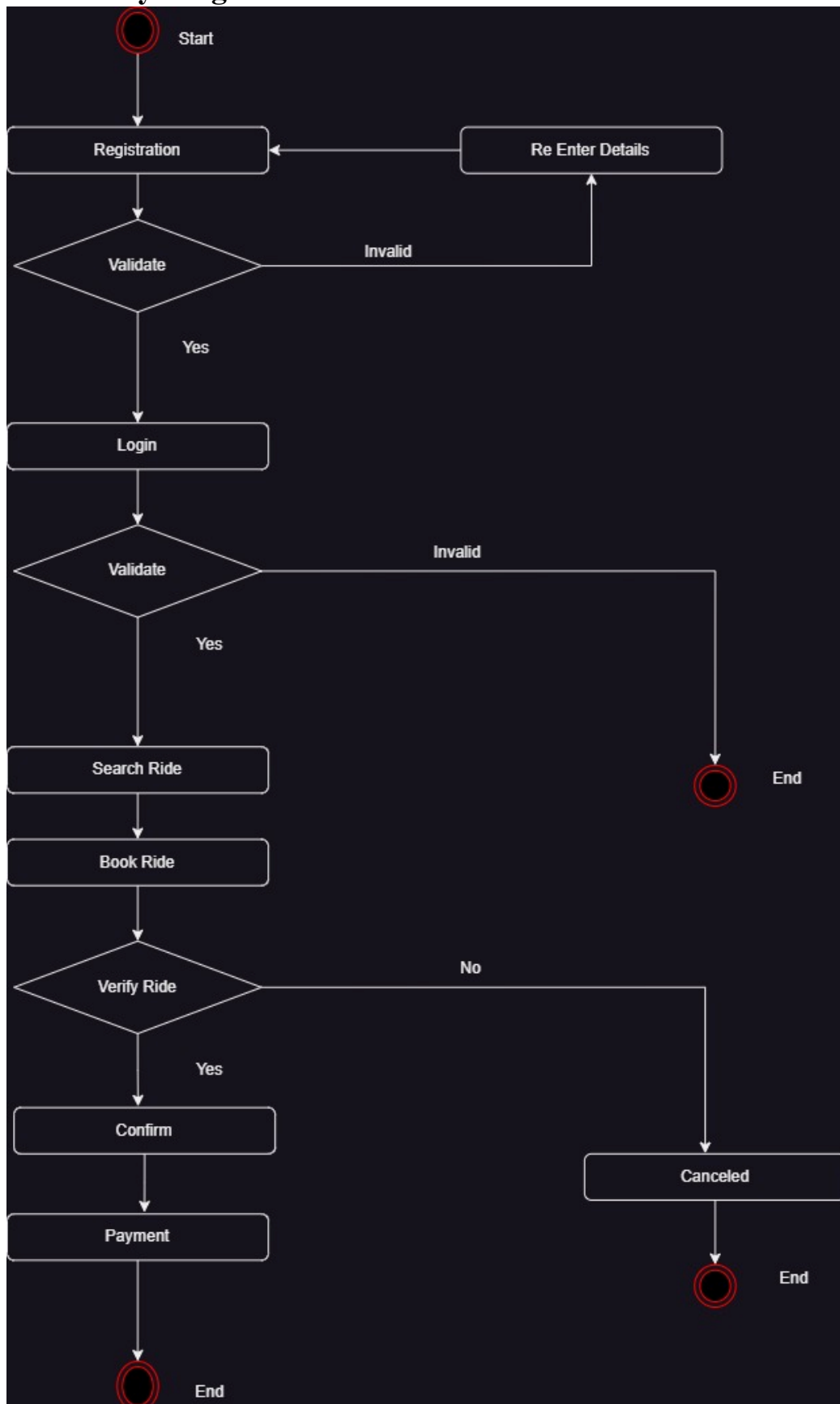
C. USE Case Owner Login

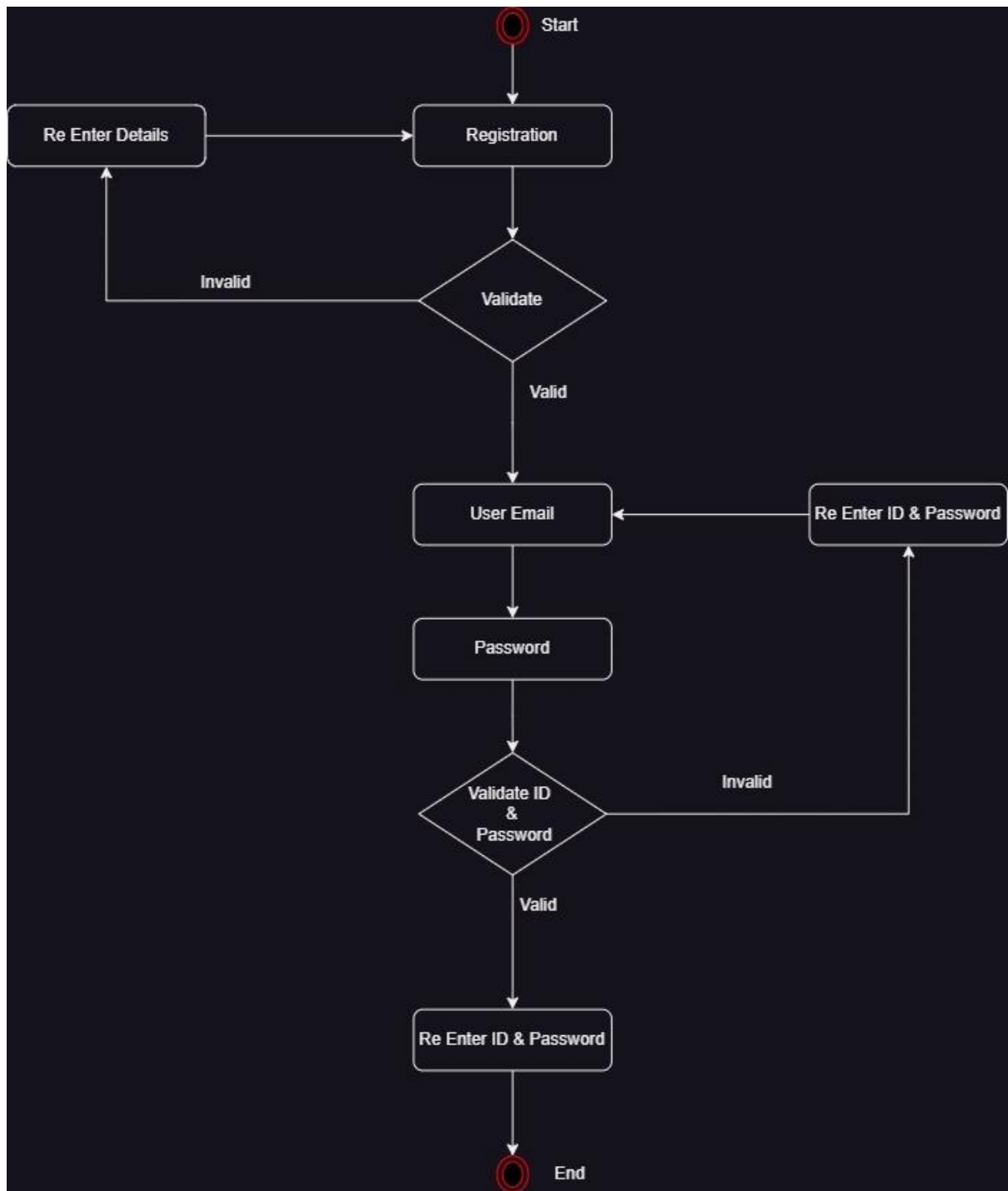## D. USE Case Customer Login Diagram
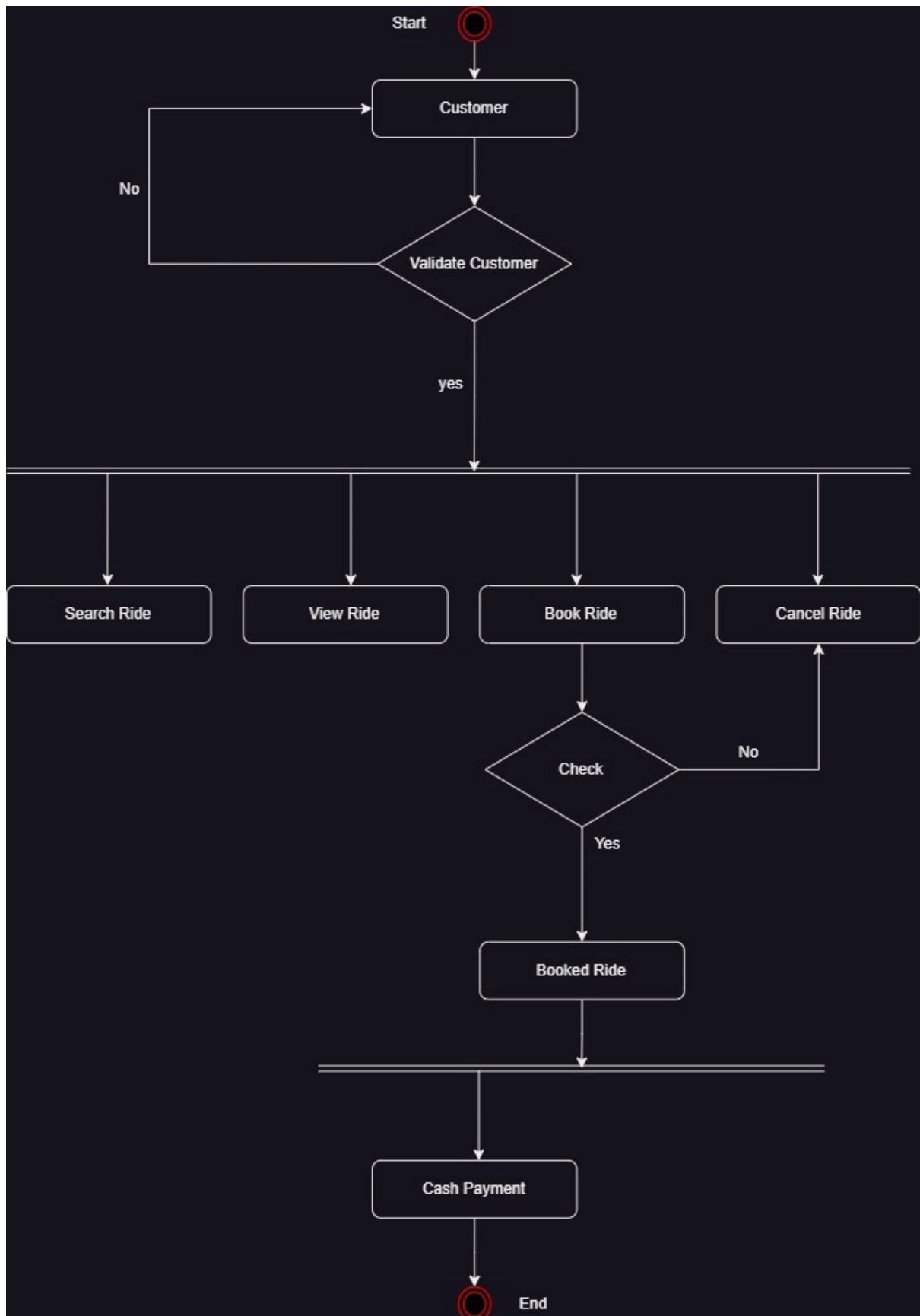
3.2 Activity Diagram

A. Activity Diagram

B. Activity Login

C. Activity Customer

3.3 Sequence Diagram

3.4 Class Diagram

Class Diagram

user

+ _id
+ name
+ email
+ password
+ phone_number
+ profile_pic
+ isVerified
+ otp

+ login ()
+ register ()

Booking

+ _id
+ user_id
+ ride_id
+ seats
+ rideData
+ createdAt
+ updatedAt
+ isCancle

+Book Ride ()
+Cancle Ride()

Ride

+ _id
+ driver_id
+ vehicel_id
+ orogin
+ destination
+ departure_time
+ price_per_seat
+ arrival_time

+Publish Ride ()
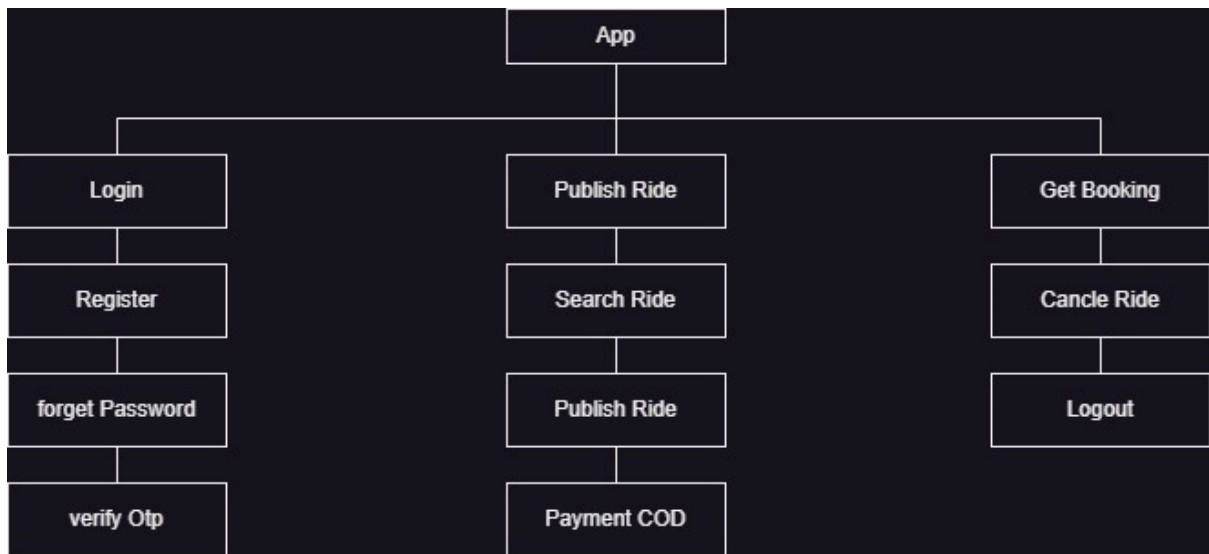+Update Ride()
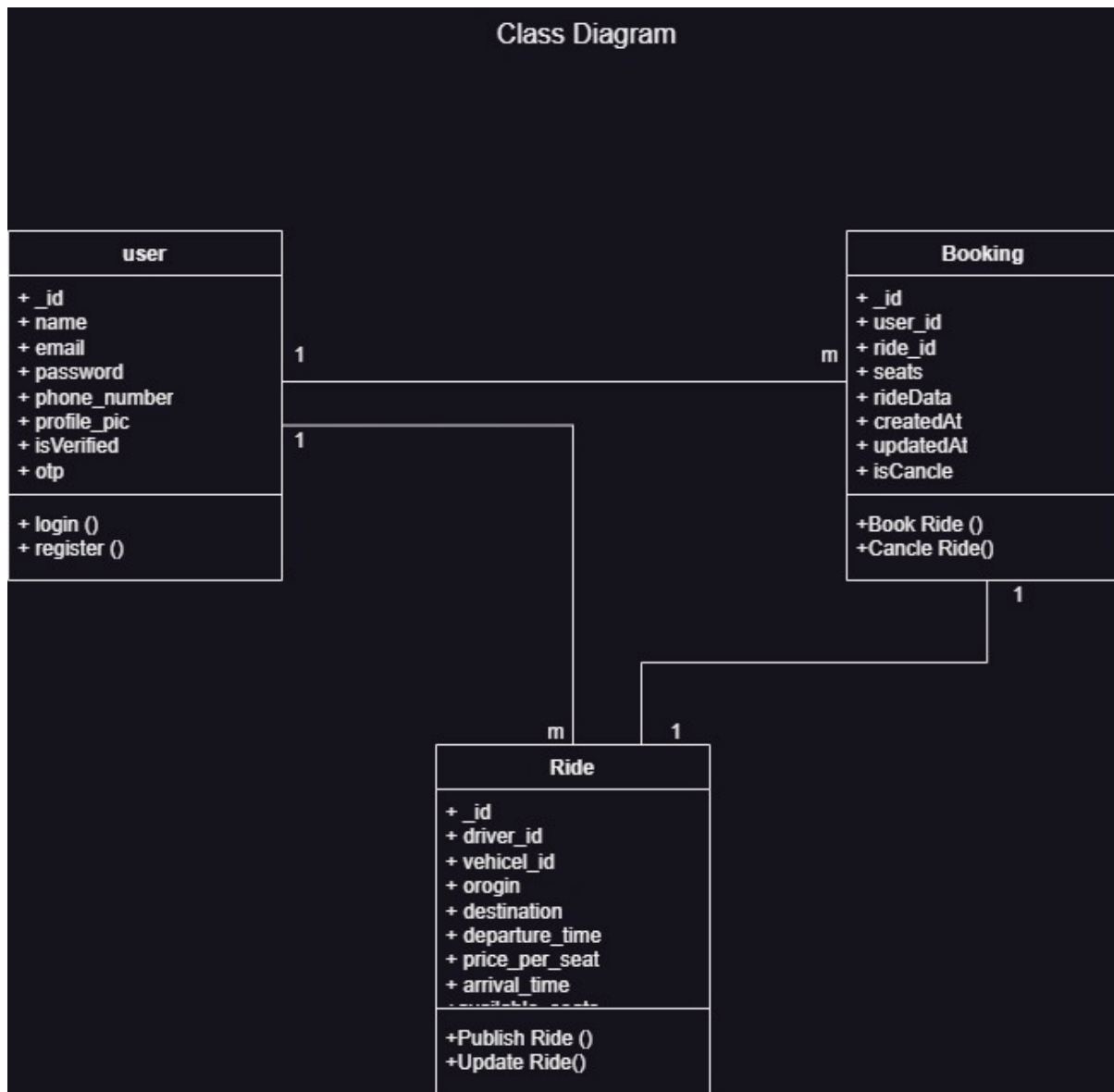
3.5 ERD (Mandatory for those using a Database)
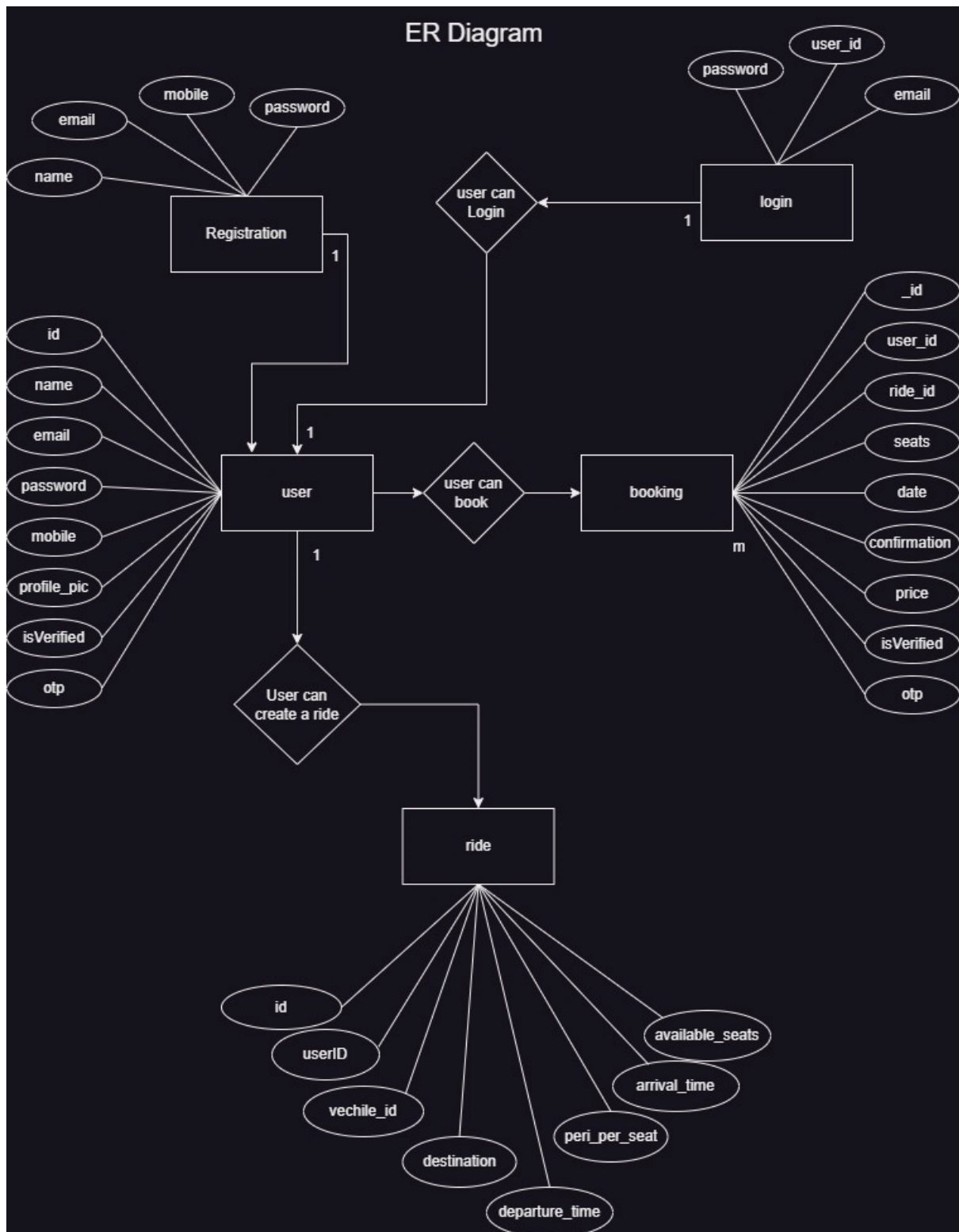
ER Diagram

3.6 Module Hierarchy Diagram

3.7 Table specifications (Database design)

A. Users

Column Name	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each user.
name	VARCHAR(100)	NOT NULL	Full name of the user.
email	VARCHAR(255)	NOT NULL, UNIQUE	User's email address. Must be unique.
password	VARCHAR(255)	NOT NULL	Encrypted password.
phone_number	VARCHAR(15)	NOT NULL	User's phone number.
profile_pic	TEXT	DEFAULT 'default-sample-pic-url'	Profile picture URL.
is_verified	BOOLEAN	DEFAULT FALSE	Indicates if the user has verified their email.
otp	VARCHAR(10)	NULL	Stores the OTP (one-time password) for verification.
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp when the record was created.
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Timestamp of the last update to the record.

B. Rides

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY	Unique identifier for the ride.
driver_id	UUID	NOT NULL, FOREIGN KEY	Reference to the driver's ID.
vehicle_id	VARCHAR(255)	NOT NULL	Identifier for the vehicle.
origin	VARCHAR(255)	NOT NULL	Start location of the ride.
destination	VARCHAR(255)	NOT NULL	End location of the ride.
departure_time	DATETIME	NOT NULL	Date and time of departure.
arrival_time	DATETIME	NOT NULL	Estimated date and time of arrival.
price_per_seat	DECIMAL(10,2)	NOT NULL	Price for a single seat.
available_seats	INTEGER	NOT NULL	Number of seats available for booking.
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp when the record was created.
updated_at	DATETIME	DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP	Timestamp when the record was last updated.

C. Bookings

Column Name	Data Type	Constraints	Description
_id	ObjectId	PRIMARY KEY	Unique identifier for the booking.
user_id	ObjectId	FOREIGN KEY (Users._id)	Reference to the user who made the booking.
ride_id	ObjectId	FOREIGN KEY (Rides._id)	Reference to the ride being booked.
seats	Integer	NOT NULL	Number of seats booked in the ride.
rideData	String (JSON)		Serialized JSON data containing ride details.
createdAt	Timestamp	DEFAULT: CURRENT_TIMESTAMP	Date and time when the booking was created.
updatedAt	Timestamp		Date and time when the booking was last updated.
__v	Integer		Version field for the document (used by MongoDB).

3.8 Data dictionary

A. Users

Field Name	Type	Description	Example Value
_id	ObjectID	A unique identifier for the user record, automatically generated by MongoDB.	"673c61622b51e7d93777db75"
name	String	The full name of the user.	"Chhagan Kumawat"
email	String	The user's email address, used for login and communication.	"chhagankumarkumawat1212@gmail.com"
password	String (Hash)	The user's password, securely stored as a hashed value.	"$2a$10$L/rABnOFL7g6B7IR28RYaOqRmc5kaKYu/YgPLGfPEQGBzovc03yry"
phone_number	String	The user's phone number, used for contact and verification.	"7232083504"
profile_pic	String (URL)	A URL or path to the user's profile picture. Default if no picture is uploaded.	"default-sample-pic-url"
isVerified	Boolean	Indicates whether the user's email/phone number has been verified.	true
otp	String/Null	A one-time password for verification purposes. Null if not in use.	null
__v	Number	Internal version key used by MongoDB for schema versioning.	0

B. Rides

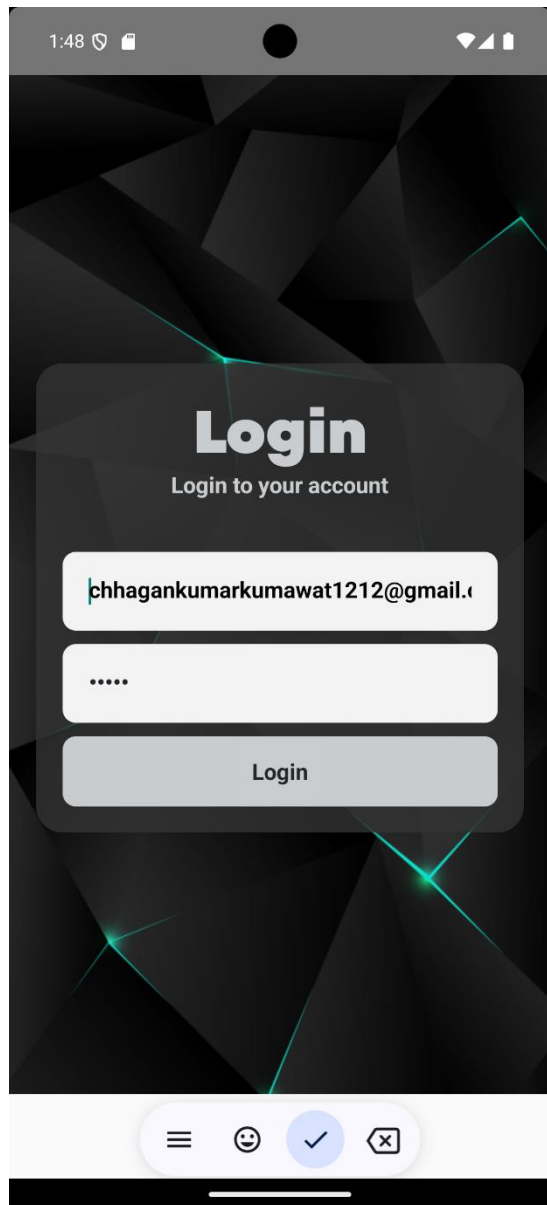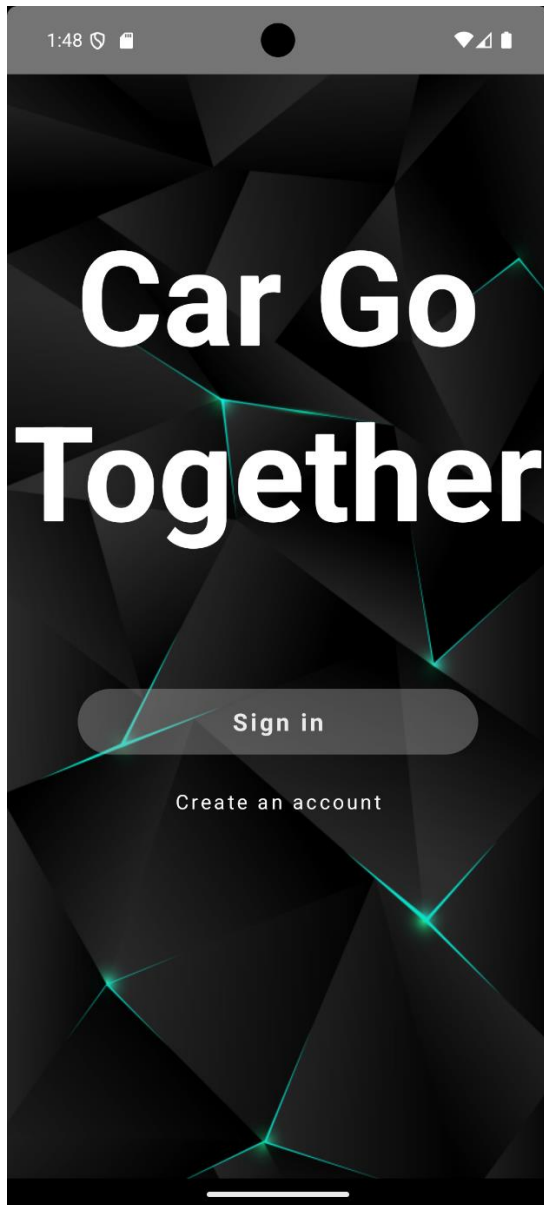| Field Name	Type	Description	Example Value
_id	Object	The unique identifier for the ride record.	{ "$oid": "67149e632002dd9f8392c17b" }
driver_id	Object	The unique identifier for the driver associated with this ride.	{ "$oid": "67149dd62002dd9f8392c175" }
vehicle_id	String	The identifier for the vehicle assigned to this ride.	"1234"
origin	String	The starting point or origin of the ride.	"pune"
destination	String	The endpoint or destination of the ride.	"rajasthan"
departure_time	Date	The date and time when the ride is scheduled to start. (ISO 8601 format)	"2024-10-20T06:07:52.698Z"
price_per_seat	Number	The cost per seat for the ride.	1212
arrival_time	Date	The date and time when the ride is scheduled to arrive at the destination. (ISO 8601 format)	"2024-10-20T08:12:00.000Z"
available_seats	Number	The number of seats available for booking on this ride.	4
__v	Number (Internal)	Version key used by MongoDB for tracking schema changes and document versions.	0

C. Bookings

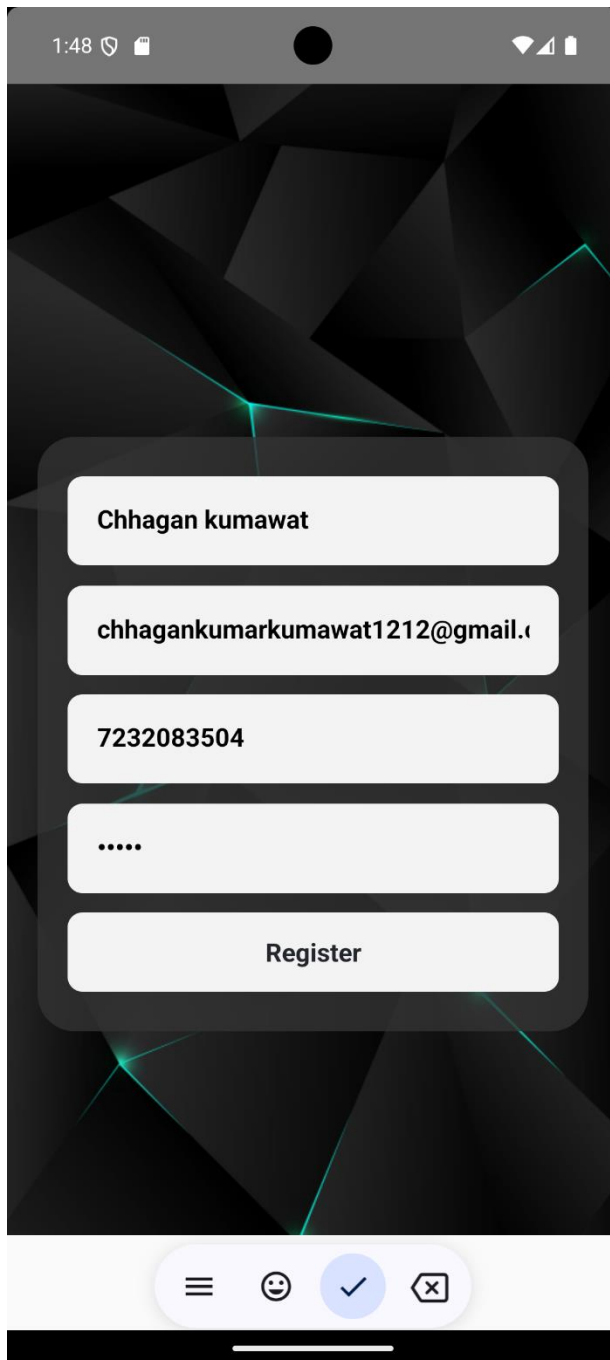| Field Name	Type	Description	Example Value
_id	ObjectId	Unique identifier for the booking record.	"6738ada10aa1d47852899548"
user_id	ObjectId	Identifier for the user who created the booking.	"6737023fc9781baba98986db"
ride_id	ObjectId	Identifier for the ride associated with this booking.	"67370f48c9781baba98986f6"
seats	Integer	Number of seats booked by the user.	1
rideData	String (JSON)	Serialized JSON string containing details about the ride (driver, origin, destination, times, pricing, etc.).	"{\"_id\":\"67370f48c9781baba98986f6\",...}"
createdAt	ISO 8601 DateTime	Timestamp when the booking record was created.	"2024-11-16T14:35:13.460Z"
updatedAt	ISO 8601 DateTime	Timestamp when the booking record was last updated.	"2024-11-16T14:35:13.460Z"
__v	Integer	Version key used by MongoDB for internal versioning of the document.	0

CHAPTER 4 : USER MANUAL

4.1 User Interface Screens (Input)

A. Home Screen & Login Screen:

B. Registration screen & OTP Generation:

C. Dashboard And Booking List:

D. Create Ride & Ride List:

12:05

Car : Audi   **16 November 2024**

Bavdhan   -----------1 hrs ----------   **Katrej**

Available Seats :1   **Price Per Seat : 120**

Car : Audi   **16 November 2024**

Bavdhan   -----------1 hrs ----------   **Katrej**

Available Seats :2   **Price Per Seat : 120**

4.2 Output Screens with data :

A. Registration Screen:

_id: ObjectId('67149dd62002dd9f8392c175')
name : "chhagan kumawat"
email : "chhagankumawat1212@gmail.com"
password : "$2a$10$XGnLUu4opXiwNonxy2zZvORF8RQN42JDel0w0bxSqZL.8Ejb.Yy8e"
phone_number : "7235698456"
profile_pic : "default-sample-pic-url"
isVerified : true
otp : null
__v : 0

_id: ObjectId('6737023fc9781baba98986db')
name : "Chhagan kumawat"
email : "chhagankumarkumawat1212@gmail.com"
password : "$2a$10$UgJBxq8TeiyOi5BZG6Y.quXTSEHUKIKqeYccxp87ZSTTu/UdpEcJe"
phone_number : "7232083504"
profile_pic : "default-sample-pic-url"
isVerified : true
otp : null
__v : 0

B. Create Ride Screen:

_id: ObjectId('67152eed316f9833ac863728')
driver_id : ObjectId('67149dd62002dd9f8392c175')
vehicle_id : "1234"
origin : "Pune"
destination : "Rajasthan"
departure_time : 2024-10-19T18:30:00.000+00:00
price_per_seat : 1400
arrival_time : 2024-10-20T06:30:00.000+00:00
available_seats : 1
__v : 0

_id: ObjectId('67370f48c9781baba98986f6')
driver_id : ObjectId('6737023fc9781baba98986db')
vehicle_id : "1234"
origin : "Bavdhan"
destination : "Katrej"
departure_time : 2024-11-15T09:04:50.755+00:00
price_per_seat : 120
arrival_time : 2024-11-15T10:16:00.000+00:00
available_seats : 1
__v : 0

_id: ObjectId('67371094c9781baba98986fd')
driver_id : ObjectId('6737023fc9781baba98986db')
vehicle_id : "1234"
origin : "Bavdhan"

4.3 Data Reports:

A. Registration Data:

_id: ObjectId('67149dd62002dd9f8392c175')
name : "chhagan kumawat"
email : "chhagankumawat1212@gmail.com"
password : "$2a$10$XGnLUu4opXiwNonxy2zZvORF8RQN42JDel0w0bxSqZL.8Ejb.Yy8e"
phone_number : "7235698456"
profile_pic : "default-sample-pic-url"
isVerified : true
otp : null
__v : 0

_id: ObjectId('6737023fc9781baba98986db')
name : "Chhagan kumawat"
email : "chhagankumarkumawat1212@gmail.com"
password : "$2a$10$UgJBxq8TeiyOi5BZG6Y.quXTSEHUKIKqeYccxp87ZSTTu/UdpEcJe"
phone_number : "7232083504"
profile_pic : "default-sample-pic-url"
isVerified : true
otp : null
__v : 0

B. Rides List:

_id: ObjectId('67152eed316f9833ac863728')
driver_id : ObjectId('67149dd62002dd9f8392c175')
vehicle_id : "1234"
origin : "Pune"
destination : "Rajasthan"
departure_time : 2024-10-19T18:30:00.000+00:00
price_per_seat : 1400
arrival_time : 2024-10-20T06:30:00.000+00:00
available_seats : 1
__v : 0

_id: ObjectId('67370f48c9781baba98986f6')
driver_id : ObjectId('6737023fc9781baba98986db')
vehicle_id : "1234"
origin : "Bavdhan"
destination : "Katrej"
departure_time : 2024-11-15T09:04:50.755+00:00
price_per_seat : 120
arrival_time : 2024-11-15T10:16:00.000+00:00
available_seats : 1
__v : 0

_id: ObjectId('67371094c9781baba98986fd')
driver_id : ObjectId('6737023fc9781baba98986db')
vehicle_id : "1234"
origin : "Bavdhan"

4.4 Test Procedures and cases:

A. User Registration Test Cases:

Test Case ID	Test Case Description	Input Data	Expected Result	Pass/Fail
TC 001	Validate empty username	Username: ""	Error message: "Username is required."	Fail
TC 002	Validate special characters	Username: "User@123"	Error message: "Username cannot contain special characters."	Fail
TC 003	Validate valid username	Username: "JohnDoe"	User registration proceeds successfully.	Pass
TC 004	Validate empty email	Email: ""	Error message: "Email is required."	Fail
TC 005	Validate invalid email format	Email: "userexample.com"	Error message: "Enter a valid email address."	Fail
TC 006	Validate valid email	Email: "chhagankumarkumawat1212@gmail.com"	User registration proceeds successfully.	Pass
TC 007	Validate empty mobile number	Mobile Number: ""	Error message: "Mobile number is required."	Fail

TC 008	Validate invalid mobile number	Mobile Number: "12345"	Error message: "Enter a valid mobile number."	Fail
TC 009	Validate valid mobile number	Mobile Number: "7232083504"	User registration proceeds successfully.	Pass
TC 010	Validate empty password	Password: ""	Error message: "Password is required."	Fail
TC 011	Validate password length	Password: "123"	Error message: "Password must be at least 8 characters long."	Fail
TC 012	Validate valid password	Password: "12345"	User registration proceeds successfully.	Pass

B. User Login Test Cases:

Test Case ID	Test Case Description	Input Data	Expected Result	Pass/Fail
TC 001	Validate empty email	Email: "", Password: "P@ssw0rd"	Error message: "Email is required."	Fail
TC 002	Validate empty password	Email: " chhagankumarkumawat1212@gmail.com ", Password: ""	Error message: "Password is required."	Fail
TC 003	Validate both fields empty	Email: "", Password: ""	Error message: "Email and Password are required."	Fail
TC 004	Validate invalid email format	Email: " chhagankumarkumawat1212@gmail.com ", Password: "P@ssw0rd"	Error message: "Enter a valid email address."	Fail
TC 005	Validate email not registered	Email: " chhagankumarkumawat1212@gmail.com ", Password: "P@ssw0rd"	Error message: "Email not found. Please register."	Fail
TC 006	Validate incorrect password	Email: " chhagankumarkumawat1212@gmail.com ", Password: "wrongpassword"	Error message: "Incorrect password. Please try again."	Fail
TC 007	Validate valid login	Email: " chhagankumarkumawat1212@gmail.com ", Password: "123456"	User successfully logged in.	Pass
TC 008	Validate case sensitivity in email	Email: " chhagankumarkumawat1212@gmail.com ", Password: "123456"	User successfully logged in (if case-	Pass

TC			insensitive match).	
TC 009	Validate SQL injection attempt	Email: " chhagankumarkumawat1212@gmail.com ", Password: "' OR '1'='1"	Error message: "Invalid credentials. "	Fail
TC 010	Validate XSS attack in email	Email: "<script>alert('XSS')</script>", Password: "P@ssw0rd"	Error message: "Invalid email address."	Fail
TC 011	Validate locked account after failures	Email: " chhagankumarkumawat1212@gmail.com ", Password: "wrongpassword" (3+ attempts)	Error message: "Account locked due to multiple failed attempts. Please reset your password."	Fail
TC 012	Validate login with inactive account	Email: " chhagankumarkumawat1212@gmail.com ", Password: "P@ssw0rd"	Error message: "Account is inactive. Please contact support."	Fail

C. OTP Test Cases :

Test Case ID	Test Case Description	Input Data	Expected Result	Pass/Fail
TC 001	Validate OTP field empty	Email: " chhagankumarkumawat1212@gmail. com ", OTP: ""	Error message: "OTP is required."	Fail
TC 002	Validate incorrect OTP	Email: " chhagankumarkumawat1212@gmail. com ", OTP: "123456"	Error message: "Invalid OTP. Please try again."	Fail
TC 003	Validate expired OTP	Email: " chhagankumarkumawat1212@gmail. com ", OTP: "789456" (expired)	Error message: "OTP has expired. Please request a new one."	Fail
TC 004	Validate valid OTP	Email: " chhagankumarkumawat1212@gmail. com ", OTP: "654321"	User successfully logs in.	Pass
TC 005	Validate OTP length	Email: "user@example.com", OTP: "12345"	Error message: "OTP must be 6 digits."	Fail
TC006	Validate special characters in OTP	Email: " chhagankumarkumawat1212@gmail. com ", OTP: "12@#45"	Error message: "OTP must contain only numbers."	Fail
TC 007	Validate multiple incorrect attempts	Email: " chhagankumarkumawat1212@gmail. com ", OTP: "111111" (3+ attempts)	Error message: "Too many incorrect attempts. OTP disabled."	Fail

TC 008	Validate OTP request for unregistered email	Email: " chhagankumAarkumawat12@gmail.com ", OTP: "123456"	Error message: "Email not found. Please register first."	Fail
TC 009	Validate OTP delivery for valid email	Email: " chhagankumarkumawat1212@gmail.com " (Request OTP)	Success message: "OTP sent to your email."	Pass
TC 010	Validate OTP delivery failure	Email: "invalidemail@example.com"	Error message: "Failed to send OTP. Check email address."	Fail
TC 011	Validate login bypassing OTP	Email: " chhagankumarkumawat1212@gmail.com ", No OTP entered	Error message: "OTP verification is required to log in."	Fail
TC 012	Validate resend OTP functionality	Email: " chhagankumarkumawat1212@gmail.com " (Request resend)	Success message: "New OTP sent to your email."	Pass

4.5 Sample program code:

A. Home page

```
import React, { useEffect } from "react";
import {
  Image,
  ImageBackground,
  StyleSheet,
  Text,
  Touchable,
  TouchableOpacity,
  View,
} from "react-native";
import { Button } from "@rneui/themed";
import bg from "../assets/images/BG-01.jpg";
import { useDispatch, useSelector } from "react-redux";
function Home(props) {
  const count = useSelector((state) => state.loginData);
  const dispatch = useDispatch();

  useEffect(() => {
    console.log(count);
  }, []);
  return (
    <View style={styles.container}>
      <ImageBackground source={bg} resizeMode="cover"
style={styles.image}>
```

```
<View
  style={{
    alignItems: "center",
    marginTop: 100,
  }}
>
  <Text
    style={{
      fontSize: 100,
      color: "#ffffff",
      fontWeight: "bold",
    }}
  >
    Car Go
  </Text>
  <Text
    style={{
      fontSize: 100,
      color: "#ffffff",
      fontWeight: "bold",
    }}
  >
    Together
  </Text>
</View>
<TouchableOpacity
```

```
  style={{
    backgroundColor: "#e2e2e254",
    width: 304,
    height: 50,
    borderRadius: 40,
    marginTop: 100,
    marginLeft: "auto",
    marginRight: "auto",
    alignItems: "center",
    justifyContent: "center",
  }}
  onPress={() => {
    props.navigation.navigate("Login");
  }}
>
  <Text
    style={{
      color: "#ffffff",
      fontSize: 18,
      letterSpacing: 2,
      fontWeight: "bold",
    }}
  >
    Sign in
  </Text>
</TouchableOpacity>
```

```
      <TouchableOpacity
        style={{
          marginTop: 25,
          marginLeft: "auto",
          marginRight: "auto",
          alignItems: "center",
          justifyContent: "center",
        }}
        onPress={() => {
          props.navigation.navigate("Register");
        }}
      >
        <Text
          style={{
            color: "#ffffff",
            fontSize: 15,
            letterSpacing: 2,
          }}
        >
          Create an account
        </Text>
      </TouchableOpacity>
    </ImageBackground>
  </View>
 );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  button: {
    flex: 1,
    marginTop: 30,
    marginLeft: "auto",
    marginRight: "auto",
    width: 304,
    borderRadius: 10,
  },
  signUp: {
    marginTop: 30,
    marginLeft: "auto",
    marginRight: "auto",
    width: 10,
    height: 50,
    backgroundColor: "rgba(5,70,82,1)",
    borderRadius: 10,
  },

  Text: {
    fontFamily: "helvetica-regular",
    fontWeight: "bold",
```

```
    color: "rgba(5,70,82,1)",

    width: 304,

    fontSize: 25,

    textAlign: "center",

    marginTop: 30,

    marginLeft: "auto",

    marginRight: "auto",

  },


  image: {

    flex: 1,

    width: "100%",

    height: "100%",

    display: "flex",

  },

});


export default Home;
```

B. Registration page:
```
import {

  View,

  Text,

  StyleSheet,

  TouchableOpacity,

  TextInput,
```

```
    ImageBackground,

    Image,

} from "react-native";

import React, { useState } from "react";

import { PostRequestCall } from "../apicall/PostReq";

import { black, lWhite, white } from "./Color";

import { useDispatch, useSelector } from "react-redux";

import { setEmailRedux } from "../redux/features/data/DataSlice";

import bg from "../assets/images/BG-01.jpg";


export default function Register(props) {
  const dispatch = useDispatch();
  const Remail = useSelector((state) => state.emailData.value);
  const [fullName, setFullName] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [password, setPassword] = useState("");
  const [toggleOTP, setToggleOTP] = useState(true);
  const [age, setAge] = useState("20");

  const [otp, setOTP] = useState(null);

  const handleRegister = async () => {
    let data = JSON.stringify({
      name: fullName,
      email: email,
```

```
    phone_number: phone,

    password: password,

  });
  console.log(data);

  dispatch(setEmailRedux(email));


  const res = await PostRequestCall("api/auth/register", data, "",
false);
  console.log(JSON.stringify(res, null, 2));

  if (res.status === 201) {

    props.navigation.navigate("verifyOTP");

  }

 };


 return (

  <View style={styles.container}>

    <ImageBackground source={bg} resizeMode="cover"
style={styles.image}>


    {/* <View

      style={{

        backgroundColor: white,

        flex: 1,

        alignItems: "center",

      }}

     > */}

      <View
```

```
style={{
  //   height: 400,
  width: "90%",
  backgroundColor: 'rgba(52, 52, 52, 0.8)',
  marginTop: 100,
  borderRadius: 25,
  padding: 20,
  // paddingTop: 10,
  // paddingBottom: 10,
  // paddingLeft: 20,
  // paddingRight: 20,
  flexDirection: "column",
  justifyContent: "space-around",
  height: 400,
}}
>
  <TextInput
    placeholderTextColor={black}
    placeholder="Full Name"
    onChangeText={(text) => {
      setFullName(text);
    }}
    style={styles.TextInput}
  ></TextInput>
  <TextInput
    placeholderTextColor={black}
```

```jsx
        placeholder="Email"
        onChangeText={(email) => {
          setEmail(email);
        }}
        style={styles.TextInput}
      ></TextInput>
      <TextInput
        placeholderTextColor={black}
        placeholder="Phone"
        onChangeText={(text) => {
          setPhone(text);
        }}
        style={styles.TextInput}
      ></TextInput>
      <TextInput
        placeholderTextColor={black}
        placeholder="Password"
        secureTextEntry={true}
        onChangeText={(text) => {
          setPassword(text);
        }}
        style={styles.TextInput}
      ></TextInput>

      <TouchableOpacity
        onPress={() => {
```

```
          handleRegister();
        }}
      >
        <Text
          style={{
            backgroundColor: white,
            padding: 15,
            borderRadius: 10,
            // marginTop: 10,
            textAlign: "center",
            fontFamily: "normalText",
            fontWeight: "bold",
            fontSize: 17,
            color: black,
          }}
        >
          Register
        </Text>
      </TouchableOpacity>
    </View>
  {/* </View> */}
  </ImageBackground>
</View>
```

C. API Call (postReq):

```javascript
import axios from 'axios';

export const PostRequestCall = async (apiPath, payload, token,
formData) => {
  let auth = {};
  if (typeof token !== 'undefined' && token !== null && token !== '') {
    auth = {
      Authorization: `Bearer ${token}`,
    };
  }
  let headers = {
    ...auth,
    Accept: 'application/json',
    'Content-Type': formData ? 'multipart/form-data' :
'application/json',
  };
  const postResponse = await axios
    .post(`http://192.168.55.70:5000/${apiPath}`, payload, {
      headers: headers,
    })
    .then(function (result) {
      return result;
    })
    .catch(err => err.response);
  return postResponse;
};
```

D. Server File:

```
// // server.js
// const express = require("express");
// const dotenv = require("dotenv");
// const cors = require("cors");
// const connectDB = require("./config/db");
// const swaggerDocs = require("./swagger");


// // Load environment variables
// dotenv.config();


// // Connect to MongoDB
// connectDB();


// const app = express();
// swaggerDocs(app);
// app.use(express.json());
// app.use(cors());


// // Routes
// app.use("/api/auth", require("./routes/authRoutes"));
// app.use("/api/rides", require("./routes/rideRoutes"));


// const PORT = process.env.PORT || 5000;
// app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

```javascript
const express = require("express");
const mongoose = require("mongoose");
const authRoutes = require("./routes/authRoutes");
const rideRoutes = require("./routes/rideRoutes");
const bookingRoutes = require("./routes/bookingRoutes");
require("dotenv").config();


const app = express();
const port = process.env.PORT || 5000;


// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));


// Routes
app.use("/api/auth", authRoutes);
app.use("/api/rides", rideRoutes); // Ensure this is correctly
configured
app.use("/api/bookings", bookingRoutes); // Ensure this is correctly
configured


// Connect to MongoDB
mongoose
  .connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
```

```
})
  .then(() => console.log("MongoDB connected"))
  .catch((err) => console.error("MongoDB connection error:", err));


// Start the server
app.listen(port, "192.168.55.70", () => {
  console.log(`Server running on port ${port}`);
});
```

4.6 Limitations and Bibliography:

Core Documentation and Resources:

1. **React Native Documentation**
   Meta Platforms, Inc.
   Official documentation for React Native, covering setup, components, APIs, and guides.
   Available at: https://reactnative.dev/docs

2. **JavaScript Documentation**
   Mozilla Developer Network (MDN).
   Comprehensive guide and reference for JavaScript, the primary language used in React Native.
   Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript

3. **React Documentation**
   Meta Platforms, Inc.
   Documentation for React, the library that underpins React Native's core principles.
   Available at: https://reactjs.org/docs

Libraries and Tools

4. **Expo Documentation**
   Expo Team.
   Reference for using Expo CLI and libraries for quick development in React Native.
   Available at: https://docs.expo.dev

5. **Redux Toolkit Documentation**
   Redux Team.
   Simplified state management for React and React Native.
   Available at: https://redux-toolkit.js.org

6. **React Navigation**
   React Navigation Team.
   Documentation for implementing navigation in React Native applications.
   Available at: https://reactnavigation.org/docs

7. **Axios Documentation**
   Axios Developers.
   Promise-based HTTP client for making API requests.
   Available at: https://axios-http.com/docs

Community Resources

9. **Stack Overflow**
   Various contributors.
   Solutions and discussions related to common React Native development challenges.
   Available at: https://stackoverflow.com

10. **GitHub Repositories**
    GitHub Inc.
    Various open-source projects and code samples for React Native.
    Examples: React Native repository (https://github.com/facebook/react-native)

11. **Blog Articles and Tutorials**
    Authors: Multiple.
    Tutorials and best practices shared by developers. Examples include blogs on Medium, Dev.to, or freeCodeCamp.

Design and UI References

12. **Material Design Guidelines**
    Google.
    Reference for implementing material design principles in React Native applications.
    Available at: https://material.io/design

13. **Figma for Mobile Design**
    Figma Team.
    Design and prototype guidelines for mobile apps.
    Available at: https://www.figma.com