Peer-to-Peer Carpooling Blockchain System

1. Introduction

Purpose

The purpose of this documentation is to provide a comprehensive guide for understanding, developing, deploying, and maintaining the peer-to-peer carpooling system utilizing blockchain technology. This documentation is intended for developers, system architects, stakeholders, and end-users who are involved in the design, implementation, and use of the system. It aims to offer detailed insights into the system's architecture, functionalities, and the underlying blockchain integration.

Overview

The peer-to-peer carpooling system is designed to facilitate efficient and secure ride-sharing among users through the use of blockchain technology. This system aims to solve common challenges in traditional carpooling platforms, such as trust issues, transaction security, and data transparency. By leveraging blockchain, the system ensures that all transactions are transparent, tamper-proof, and verifiable.

Key features of the system include:

- **Decentralized Network**: Operates on a decentralized blockchain network to ensure trust and transparency.
- **Smart Contracts**: Utilizes smart contracts to automate and secure transactions between car owners and riders.
- **User Authentication**: Secure user authentication to ensure that only verified users can participate in the carpooling process.
- **Ride Matching**: Advanced algorithms to match riders with car owners based on location, time, and preferences.
- **Payment Processing**: Secure and transparent payment processing using cryptocurrency or fiat currency through blockchain transactions

Scope

This documentation covers the following aspects of the peer-to-peer carpooling system:

- System Overview: An in-depth look at the system's architecture and components.
- Functional Requirements: Detailed descriptions of the system's functionalities and user roles.
- Non-Functional Requirements: Performance, security, scalability, and reliability requirements.
- Blockchain Integration: Explanation of how blockchain technology is integrated into the system, including smart contracts and transaction flows.
- System Design: Data models, API descriptions, and user interface design.
- Conclusion

2. System Overview

System Components

The peer-to-peer carpooling system is composed of several key components, each playing a critical role in the overall functionality and performance of the platform.

1. **User Interface (UI)**
    - **Description**: The UI is the front-end component that allows users (car owners and riders) to interact with the carpooling system. It provides features such as ride search, ride booking, user registration, payment processing, and reviews.
    - **Technologies**: HTML, CSS, JavaScript, React.js (or any front-end framework you prefer).

2. **Blockchain Network**
    - **Description**: This is the decentralized network where all transactions and data are recorded. It ensures transparency, security, and immutability of the data. The blockchain network handles the storage of transaction records, user identities, and smart contract executions.
    - **Technologies**: Ethereum (or any other blockchain platform like Hyperledger, Binance Smart Chain), Solidity (for writing smart contracts).
3. **Smart Contracts**
    - **Description**: Smart contracts are self-executing contracts with the terms of the agreement directly written into code. They automate and enforce the terms of carpooling agreements between car owners and riders, including payment transactions, ride confirmations, and dispute resolutions.
    - **Technologies**: Solidity (for Ethereum smart contracts), VS Code IDE (for smart contract development and testing).
4. **Backend Server**
    - **Description**: The backend server handles the business logic, processes requests from the UI, interacts with the blockchain network, and manages data storage for non-blockchain data. It acts as a bridge between the UI and the blockchain network.
    - **Technologies**: Node.js .
5. **Ride Matching Algorithm**
    - **Description**: This algorithm is responsible for matching riders with car owners based on various criteria such as location, time, preferences, and past ratings. It ensures that users are paired efficiently and fairly.
    - **Technologies**: Solidity (for algorithm development).

6. **Payment Gateway**
    - **Description**: The payment gateway facilitates secure transactions between car owners and riders. It can handle both cryptocurrency payments (via blockchain) and fiat currency payments (via traditional payment processors).
    - **Technologies**: Ethereum (for cryptocurrency payments).

Technologies Used

The system leverages a variety of technologies and tools to ensure efficient operation, scalability, and security.

1. **Blockchain Platform**
    - **Ethereum**: Used for deploying smart contracts and handling cryptocurrency transactions. Provides a decentralized and tamper-proof ledger for all transactions.
2. **Programming Languages**
    - **Solidity**: For writing smart contracts on the Ethereum blockchain.
    - **JavaScript**: For front-end development and backend server scripting.
3. **Frameworks and Libraries**
    - **React.js**: For building the user interface.
    - **Node.js and Express.js**: For backend server development.
    - **Web3.js**: For interacting with the Ethereum blockchain from the frontend.
4. **Development Tools**
    - **VS Code IDE**: For developing and testing Solidity smart contracts.
    - **Ganache**: For local blockchain development and testing.
    - **Truffle**: For smart contract deployment and management.
    - **Git**: For version control and collaboration.

By utilizing these components and technologies, the peer-to-peer carpooling system ensures a secure, transparent, and efficient platform for users to share rides.

3. Functional Requirements

User Roles

- **Car Owner**
    - Registers and verifies their identity.
    - Lists available rides, including details such as time, date, starting location, and destination.
    - Manages ride requests from riders.
    - Receives payments for completed rides.
- **Rider**
    - Registers and verifies their identity.
    - Searches for available rides based on location and time.
    - Sends ride requests to car owners.
    - Makes payments for the ride.
- **Administrator**
    - Manages user accounts and verifies identities.
    - Monitors system activity and ensures compliance with policies.
    - Handles disputes between car owners and riders.
    - Updates and maintains the system.

Use Cases

1. **User Registration**
    - Users (car owners and riders) register and create accounts.
    - Identity verification through documentation.
2. **Ride Listing**
    - Car owners list available rides with all necessary details.
    - Option to edit or delete listed rides.

3. **Ride Search and Request**
    - Riders search for rides based on criteria like location, date, and time.
    - Riders send requests to car owners.
    - Car owners accept or decline ride requests.
4. **Payment Processing**
    - Secure payment transaction initiated once the ride request is accepted.
    - Use of cryptocurrency or fiat currency for payments.
    - Payment confirmation recorded on the blockchain.
5. **Ride Confirmation**
    - Details of the ride confirmed and shared with both car owner and rider.
    - Real-time notifications about ride status.

Features

1. **Ride Matching**
    - Advanced algorithms to match riders with car owners based on location, time, and preferences.
    - Consideration of user ratings and past interactions.
2. **Payment Processing**
    - Integration with payment gateways for secure transactions.
    - Blockchain-based payment verification and recording.
3. **User Authentication**
    - Robust authentication mechanisms to verify user identities.
    - Use of smart contracts to automate identity verification processes.
4. **Real-time Notifications**
    - Instant notifications for ride requests, confirmations, and updates.
    - Notifications for payment status and ride completion.

4. Non-Functional Requirements

Performance

- **Response Time**: The system should respond to user requests within 2 seconds.
- **Throughput**: The system should handle at least 1000 concurrent users without performance degradation.
- **Latency**: Payment processing latency should be less than 5 seconds.

Security

- **Data Encryption**: All user data and transactions should be encrypted using industry-standard encryption protocols.
- **Access Control**: Role-based access control to restrict access to sensitive data and functionalities.
- **Smart Contract Security**: Thorough auditing of smart contracts to prevent vulnerabilities and exploits.

Scalability

- **Horizontal Scaling**: The system should support horizontal scaling to handle increasing numbers of users and transactions.
- **Load Balancing**: Implementation of load balancing to distribute traffic evenly across servers.

Reliability

- **Backup and Recovery**: Regular backups and a robust recovery plan in case of system failures.
- **Error Handling**: Comprehensive error handling mechanisms to ensure smooth recovery from unexpected issues

By addressing these functional and non-functional requirements, the peer-to-peer carpooling system will be able to deliver a secure, efficient, and user-friendly experience for all participants.

5. Blockchain Integration

Smart Contracts

The system uses two main smart contracts: **RideContract** and **Migrations**.

1. **RideContract**: This contract manages the entire ride-sharing process, including ride creation, joining rides, handling passenger lists, and managing balances. Here are the key functions and structures:
    - **Structures**:
        - **Ride**: Stores information about each ride such as the driver, start and end points, fare, start time, number of seats, passengers, and status.
        - **Passenger**: Stores information about passengers including address, phone number, number of people, verification code, and arrival status.
        - **BalanceChange**: Tracks balance changes for users with details like the user affected, amount changed, balances before and after the change, timestamp, and a description.
    - **Functions**:
        - **createRide**: Allows drivers to create a new ride.
        - **joinPendingRide**: Allows passengers to join a ride and enter a pending state awaiting driver approval.
        - **acceptPassenger**: Drivers can accept pending passengers, moving them to the confirmed passengers list.
        - **declinePassenger**: Drivers can decline pending passengers and refund their fare.
        - **cancelRide**: Allows passengers to cancel their pending ride and receive a refund.
        - **completeRide**: Marks a ride as completed, transferring the fare to the driver and refunding non-arrived passengers.

        - **arrive**: Passengers can confirm their arrival.
        - Helper functions like **getCreatedRides**, **getJoinedRides**, **getRideDetails**, and balance management functions.
    - **Events**:
        - Various events like **RideCreated**, **PassengerJoined**, **PassengerAccepted**, **PassengerDeclined**, **RideCompleted**, and **BalanceChanged** to notify about state changes.
2. **Migrations**:
    - This contract is used to manage the deployment of smart contracts, ensuring that they are deployed in a specific order and tracking the last completed migration.

Blockchain Network

The described system can be deployed on either a public or private blockchain network, depending on the specific use case and requirements.

- **Public Blockchain**:
    - Ethereum mainnet or testnets (Ropsten, Rinkeby, Goerli).
    - Benefits: Decentralization, security, and widespread adoption.
    - Drawbacks: Higher transaction fees, slower transaction times.
- **Private Blockchain**:
    - Custom networks using frameworks like Hyperledger Fabric or Ethereum (using tools like Geth or Quorum).
    - Benefits: Lower costs, faster transactions, greater control over the network.
    - Drawbacks: Centralization risks, potential security concerns.

The chosen consensus mechanism also depends on the network type:

- **Proof of Work (PoW)**: Used by Ethereum mainnet.
- **Proof of Stake (PoS)**: Transitioning with Ethereum 2.0.

- **Practical Byzantine Fault Tolerance (PBFT)** or other consensus mechanisms for private networks.

Transaction Flow

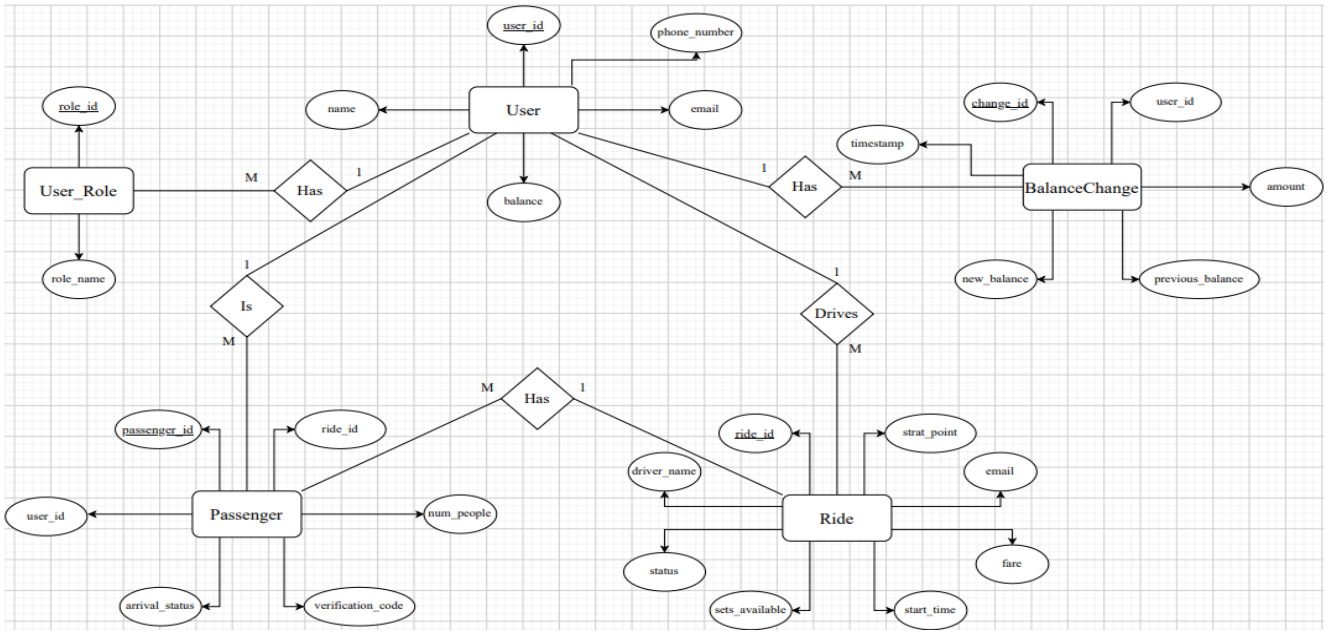
1. **Ride Creation**:
    - The driver creates a ride using **createRide**.
    - The **RideCreated** event is emitted, recording the ride on the blockchain.
2. **Passenger Joining**:
    - Passengers join rides using **joinPendingRide**, making a payment for the fare.
    - The **PassengerJoined** event is emitted.
    - The passenger's balance is adjusted, and a **BalanceChanged** event is emitted.
3. **Driver Actions**:
    - The driver can accept passengers using **acceptPassenger**, emitting the **PassengerAccepted** event.
    - Alternatively, the driver can decline passengers using **declinePassenger**, refunding the fare and emitting the **PassengerDeclined** event.
4. **Ride Cancellation**:
    - Passengers can cancel their pending status using **cancelRide**, receiving a refund and emitting the **PassengerCancelled** event.
5. **Ride Completion**:
    - Once the ride is completed, the driver calls **completeRide**.
    - The **RideCompleted** event is emitted.
    - The driver receives the total fare for arrived passengers, and non-arrived passengers are refunded.
    - Balance changes are recorded and emitted via **BalanceChanged**.

6. **Passenger Arrival**:
    - Passengers confirm their arrival using **arrive**, emitting the **PassengerArrived** event.

Each transaction on the blockchain involves interactions with these smart contracts, ensuring transparency, immutability, and security of ride-sharing operations. The use of events helps in tracking the state changes and providing an audit trail for all transactions.

6. System Design

ERD



7. Conclusion

In conclusion, this documentation provides a comprehensive overview of our peer-to-peer carpooling system using blockchain. By leveraging blockchain technology, we have created a secure, transparent, and efficient platform for users to share rides and reduce their carbon footprint. The system's architecture, smart contracts, and user interface have been carefully designed to provide a seamless experience for both car owners and riders. With further enhancements and continuous maintenance, we aim to improve the system's functionality and scalability, ensuring it remains a sustainable solution for modern transportation needs.