International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 7 July 2023, pp: 858-869 www.ijaem.net ISSN: 2395-5252

Peer-to-Peer Carpooling D App

[1]Shaik Irfan Babu, [2]Akash Belide, [3]Harshith Reddy
[1,2]Assistant professor, Department of CSE, Mahatma Gandhi Institute of Technology[2,3]Student, Department of CSE, Mahatma Gandhi Institute of Technology

Date of Submission: 20-07-2023 Date of Acceptance: 31-07-2023

ABSTRACT -The car-sharing market is currently witnessing exponential growth, surpassing conventional car ownership and garnering significant attention from Original Equipment Manufacturers (OEMs). However, the widespread adoption of centralized ride-sharing platforms, exemplified by companies like Uber and Ola, has raised concerns regarding data security. These platforms store extensive data of both drivers and riders, making them vulnerable to potential hacker attacks and password leaks. Moreover, traditional car-sharing systems that involve third-party intermediaries can lead to privacy breaches and misuse of customer data.

In response to these challenges, this research paper proposes a novel approach based on blockchain technology to establish a more secure and transparent ecosystem for carpooling services. By leveraging blockchain's decentralized and immutable characteristics, the research aims to create a peer-to-peer carpooling application using smart contracts, which would directly connect riders and drivers, eliminating the need for intermediaries. The application will be built using Solidity, a programming language for Ethereum-based smart contracts.

Key words: Blockchain, Decentralization, Smart Contracts, Data Security.

I. INTRODUCTION

The car-sharing market has witnessed a remarkable surge in popularity, offering a cost-effective and convenient mode of transportation for users. However, the traditional centralized car-sharing model introduces significant challenges, particularly concerning the security and privacy of users' sensitive data. Additionally, the presence of intermediaries such as Uber and Ola adds complexity to the system, hindering direct interactions between users and drivers and incurring additional commission fees.

To address these pressing concerns and drive transformative change, this research paper proposes an innovative solution: the decentralization of car-sharing through the integration of blockchain technology and smart contracts. By harnessing the inherent advantages of blockchain, our primary objective is to establish a more secure, transparent, and user-centric car-sharing ecosystem. In this decentralized framework, data security is prioritized, fostering trust and confidence among all participants. Moreover, this paradigm shift empowers users and drivers to directly engage with each other, eliminating the need for intermediaries and creating a fair and equitable platform for all stakeholders.

In the following sections, we will present a detailed overview of the proposed decentralized car-sharing model, the underlying blockchain technology, and the implementation of smart contracts. Additionally, we will show the working of the proposed model. By the end of this research, we envision a more comprehensive understanding of the benefits andpotential associated with the model to reshape the future of transportation.

1.1 PROBLEM STATEMENT

Currently, traditional car-sharing systems are centralized, and there are hundreds and thousands of vehicles in India that travel from one city to another, passing through neighborhoods to metropolitan areas daily. However, the number of taxis in major cities falls significantly short of the demand, leaving many people waiting for a taxi by the roadside.

Additionally, the increasing number of cars is causing adverse environmental consequences, such as car exhaust emissions, a wide variety of gases, and hazardous materials, which can lead to global warming and harm human health. Furthermore, cab and car drivers who are not covered by any taxi-hailing service face dramatically increased unemployment rates. Moreover, due to the attractive prices offered by Ola and Uber, local drivers have lost a substantial portion of their customer base. Lastly, ride service networks also encounter issues related to user privacy, including the security of financial details, the privacy of personal information, and location data.

1.2 EXISTING SYSTEM

A significant obstacle confronted by corporations is the safeguarding of confidential data concerning drivers and passengers, with the primary focus being on security and privacy. Despite substantial financial investments in user authentication, incidents of fraudulent identities continue to rise on a daily basis

1.3 PROPOSED SYSTEM

The proposed system is a decentralized car-sharing network that addresses security and privacy concerns while empowering individuals to generate income. Leveraging blockchain and smart contracts, the system ensures data security and eliminates intermediaries like Uber and Ola. Drivers with modern, secure vehicles and a smartphone or computer can directly offer car-sharing services to riders, earning income without intermediary commissions. The system prioritizes user authentication and fraud prevention and aims to create an accessible and inclusive platform for car owners. This innovative approach presents opportunities to revolutionize the car-sharing industry and promote environmental sustainability.

II. LITERATURE SURVEY

The paragraphs provided discuss various research and development efforts aimed at improving car-sharing systems using blockchain.

In [1], Car-sharing systems provide shared vehicles to people, reducing the reliance on personal vehicles and addressing urban challenges. However, the existing car-sharing system faces security issues as sensitive information like user identity, location data, and access codes are transmitted through public channels, leaving them vulnerable to attackers. To enhance security and address centralization concerns, this study proposes a decentralized car-sharing scheme using blockchain technology. The system ensures data integrity, enables anonymous authentication of participants, and offers resistance against various attacks. The scheme is validated through informal analysis, AVISPA simulation, and BAN logic analysis, while computation and communication costs are analyzed.

In [2], Shared electric and automated mobility (SEAM) is an essential feature of urban mobility in smart cities. This article emphasizes cybersecurity for SEAM, particularly for peer-to-peer car-sharing, and presents security solutions based on conjugated authentication and authorization.

In [3], The shared mobility concept is disruptive and transformative for the automotive industry, providing alternatives to car ownership through e-hailing, car-sharing, and other solutions. This paper explores how blockchain and IoT technologies can drive shared mobility forward. It presents a high-level architecture for a blockchain-IoT-based platform, combining car-sharing and car-leasing, streamlining processes for stakeholders. The design considers security, privacy, authenticity, traceability, reliability, scalability, and interoperability in car-sharing platforms.

In [4], Interconnected smart vehicles offer sophisticated services but expose vehicles to security and privacy threats. The article proposes a blockchain-based architecture to protect user privacy and enhance the security of the vehicular ecosystem. The proposed architecture is illustrated through wireless remote software updates and dynamic vehicle insurance fees, demonstrating resilience against common security attacks.

Lastly In [5], This work focuses on the formal semantics of Solidity, Ethereum's Turing-complete programming language for smart contracts. The paper defines the semantics of Solidity to provide a formal specification of smart contracts and establish semantic-level security properties for high-level verification. The proposed semantics ensure correct and secure execution behaviors of smart contracts to assist developers in writing secure code and reasoning about compiler bugs.

IV. DESIGN METHODOLOGY

This section deals with the design implementations of the system. These include a user flow diagram for booking a cab, block diagram and system architecture. Such designs give an overview of how the system functions and important aspects of the project.

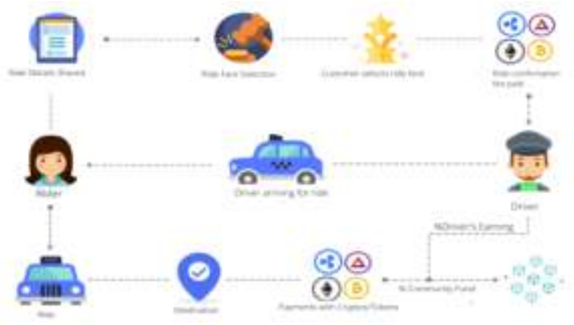


Figure. 3.1 User flow for booking a cab

In Figure 3.1, Illustrates the proposed system design, this diagram shows the user flow for booking a cab. It's almost similar to the existing flow, with additional processes such as selection of best ride and crypto payments being performed.

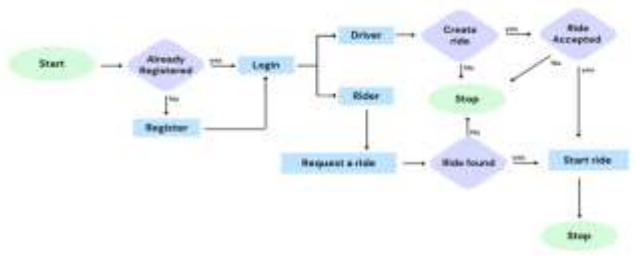


Figure. 3.2 block diagram illustrating user flow for booking a ride

Figure 3.2, Shows the block diagram illustrating user flow for booking a ride, this diagram shows the user flow for booking a cab. It's almost similar to the existing flow, with additional processes such as selection of best ride and crypto payments being performed.

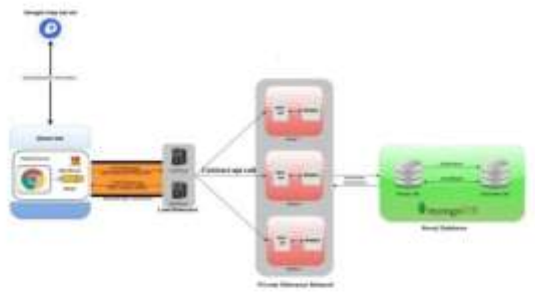


Figure. 3.3 Architecture diagram

Figure 3.3, Shows the architecture diagram. Here, the web application is developed using MERN (MongoDB, Express, React, and Nodejs) software stack and to store the data, MongoDB, which is a NoSQL database. To facilitate the smooth and seamless development of decentralized apps, we used an Ethereum Blockchain-based development environment. A private Ethereum Blockchain is set up using Ganache which is a hosted Ethereum Node cluster. The Google Map APIs are integrated to access navigation services. The current trip status is stored in the JSON format and displayed on the user dashboard.

V. IMPLIMENTATION

This section provides a detailed overview of the implementation of the system. It discusses the development environment, programming languages, and frameworks used in the system. The section also describes the key functionalities implemented, including user registration, ride requests, fare calculations, and payment processing. Furthermore, it includes code snippets, examples, and additional relevant information specific to the implementation of the proposed system.

The project encompasses both the back-end smart contract development and the front-end user interface implementation. Here are the details of the development environment and tools used for each aspect:

Smart Contract Development: For smart contract development, we utilized tools such as Ganache and Truffle. Ganache allows for the creation of a local blockchain network, which is crucial for testing and deploying smart contracts locally. Truffle, on the other hand, offers a development framework that simplifies the process of writing and deploying smart contracts. Truffle provides utilities for compiling contracts, managing migrations, and interacting with deployed contracts during development.

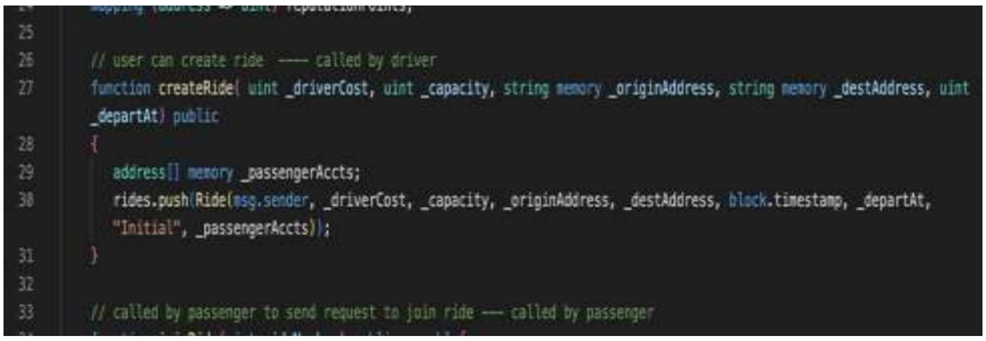
Front-end Development: The front-end of the project is built using React, a popular JavaScript library for building user interfaces. React offers a component-based approach, making it easier to develop reusable and maintainable UI components. To set up the React development environment, tools such as Node.js and npm (Node Package Manager) are used. Additional libraries and frameworks, such as React Router for navigation and state management libraries like Redux or React Context, are incorporated .

Integrated Development Environment (IDE): Visual Studio Code (VS Code) is used for both smart contract and front-end

development. VS Code provides a range of extensions that enhance the development experience, including Solidity extensions for smart contract development and React-related extensions for front-end development. It offers features like syntax highlighting, code completion, and debugging capabilities, aiding developers in writing efficient and error-free code.

By utilizing a combination of Ganache and Truffle for smart contract development and React with tools like Create React App and VS Code for front-end development, this covers both the blockchain back-end and the user interface front-end.

Ride Creation: The contract allows users to create new ride offers by specifying the necessary details such as the pickup and drop-off locations, date, time, and fare.

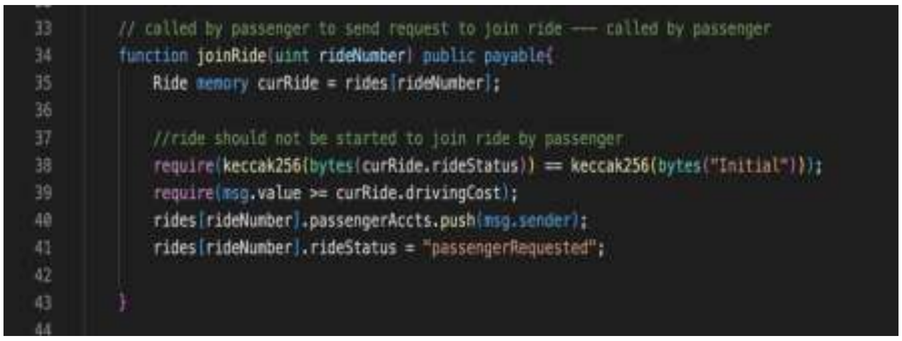


```
// user can create ride  ---- called by driver
function createRide( uint _driverCost, uint _capacity, string memory _originAddress, string memory _destAddress, uint
_departAt) public
{
  address[] memory _passengerAccts;
  rides.push(Ride(msg.sender, _driverCost, _capacity, _originAddress, _destAddress, block.timestamp, _departAt,
  "Initial", _passengerAccts));
}

// called by passenger to send request to join ride --- called by passenger
```

Figure 4.1 code snippet of Ride Creation function

Figure 4.1, Shows the function which allows users to create a new ride offer by providing the necessary details such as the pickup and drop-off locations, date, time, and fare. The function emits an event to notify the front-end and other participants about the newly created ride.

Ride Joining: Users can join existing rides by submitting a request with the required details such as the ride ID and desired number of seats.

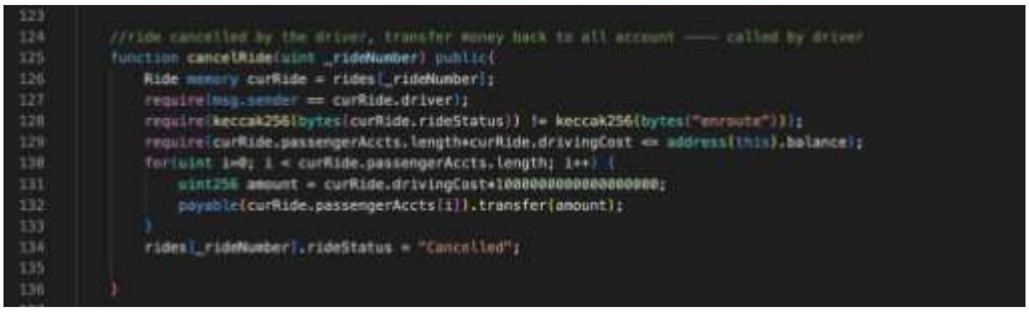


```
// called by passenger to send request to join ride --- called by passenger
function joinRide(uint rideNumber) public payable{
    Ride memory curRide = rides[rideNumber];

    //ride should not be started to join ride by passenger
    require(keccak256(bytes(curRide.rideStatus)) == keccak256(bytes("Initial")));
    require(msg.value >= curRide.drivingCost);
    rides[rideNumber].passengerAccts.push(msg.sender);
    rides[rideNumber].rideStatus = "passengerRequested";

}
```

Figure 4.2 code snippet of Ride Joining function.

Figure 4.2, Shows the function which allows users to join an existing ride by submitting a request with the required details, such as the ride ID and desired number of seats. The function validates the availability of seats and emits an event to notify the front-end and the ride creator about the new participant.

Cancel Ride: The smart contract allows driver to cancel the ride. If the ride is not en route, the driver can cancel the ride

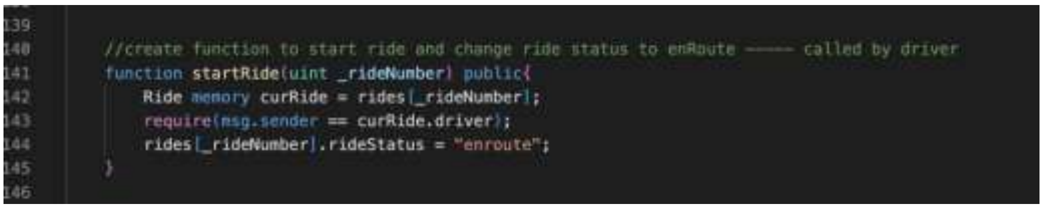


```
//ride cancelled by the driver, transfer money back to all account ---- called by driver
function cancelRide(uint _rideNumber) public{
    Ride memory curRide = rides[_rideNumber];
    require(msg.sender == curRide.driver);
    require(keccak256(bytes(curRide.rideStatus)) != keccak256(bytes("enroute")));
    require(curRide.passengerAccts.length*curRide.drivingCost <= address(this).balance);
    for(uint i=0; i < curRide.passengerAccts.length; i++) {
        uint256 amount = curRide.drivingCost*1000000000000000000;
        payable(curRide.passengerAccts[i]).transfer(amount);
    }
    rides[_rideNumber].rideStatus = "Cancelled";

}
```

Figure 4.3 code snippet of Cancel Ride function.

Figure 4.3, Shows the function which allows the driver to cancel a ride. It takes the ride number as input and verifies that the caller is the driver, and the ride is not en route. If the conditions are met, it transfers the driving cost back to all passengers by iterating through the **passengerAccts** array and using the **transfer** function to send the appropriate amount to each passenger. Finally, it updates the ride status to "Cancelled."

Start Ride: Allows the driver to start a ride, changing the ride status to “enroute”.

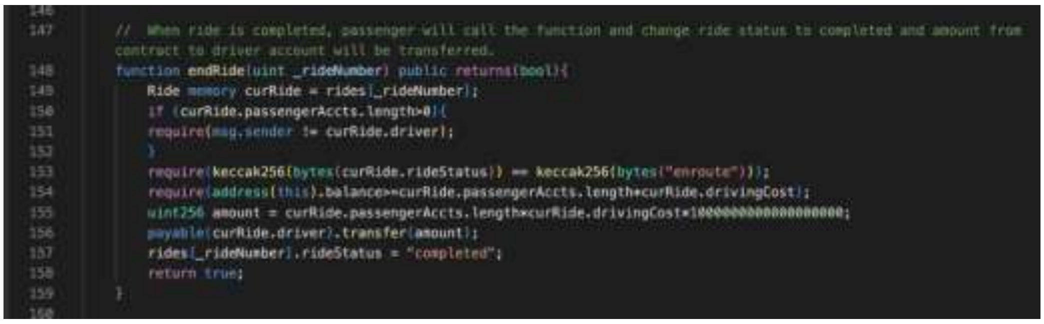


```
//create function to start ride and change ride status to enRoute ----- called by driver
function startRide(uint _rideNumber) public{
    Ride memory curRide = rides[_rideNumber];
    require(msg.sender == curRide.driver);
    rides[_rideNumber].rideStatus = "enroute";
}
```

Figure 4.4 code snippet of Start Ride function.

Figure 4.4, Shows the function which allows the driver to start a ride. It takes the ride number as input and verifies that the caller is the driver. If the conditions are met, it updates the ride status to "enroute."

End Ride: Once a ride is completed, the smart contract facilitates the finalization of the ride, ensuring the accurate distribution of funds among the participants.

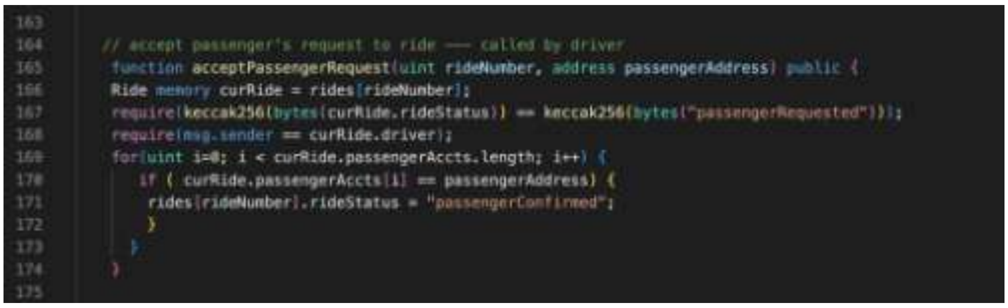


```
//  When ride is completed, passenger will call the function and change ride status to completed and amount from
contract to driver account will be transferred.
function endRide(uint _rideNumber) public returns(bool){
    Ride memory curRide = rides[_rideNumber];
    if (curRide.passengerAccts.length>0){
    require(msg.sender != curRide.driver);
    }
    require(keccak256(bytes(curRide.rideStatus)) == keccak256(bytes("enroute")));
    require(address(this).balance>=curRide.passengerAccts.length*curRide.drivingCost);
    uint256 amount = curRide.passengerAccts.length*curRide.drivingCost*1000000000000000000;
    payable(curRide.driver).transfer(amount);
    rides[_rideNumber].rideStatus = "completed";
    return true;
}
```

Figure 4.5 code snippet of End Ride function.

Figure 4.5, Shows the function which allows a passenger (other than the driver) to end the ride. It takes the ride number as input and verifies that the caller is not the driver, the ride status is "enroute," and the contract balance is sufficient to cover the driving cost for all passengers. If the conditions are met, it transfers the driving cost from each passenger's balance to the driver'saddress, updates the ride status to "completed," and distributes reputation points among the participants.

Accept Passenger Request: Allows the driver to accept a passenger's request to join the ride, changing the ride status to "passengerConfirmed".

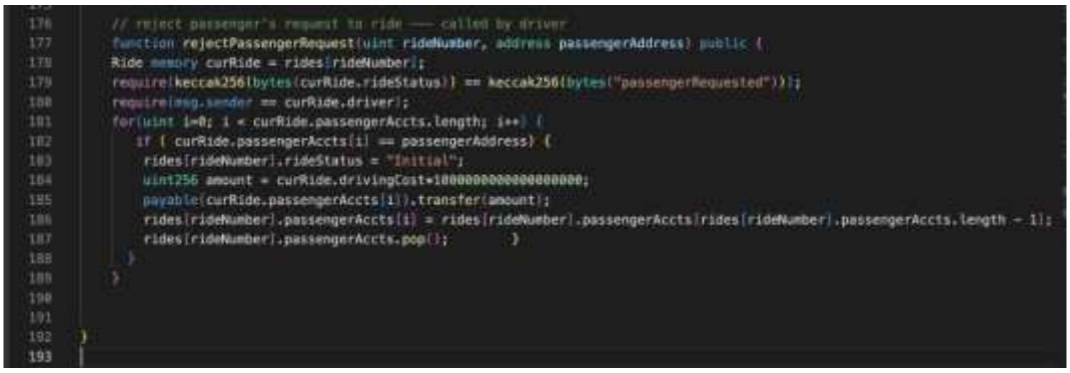


```
// accept passenger's request to ride —— called by driver
 function acceptPassengerRequest(uint rideNumber, address passengerAddress) public {
 Ride memory curRide = rides[rideNumber];
 require(keccak256(bytes(curRide.rideStatus)) == keccak256(bytes("passengerRequested")));
 require(msg.sender == curRide.driver);
 for(uint i=0; i < curRide.passengerAccts.length; i++) {
    if ( curRide.passengerAccts[i] == passengerAddress) {
     rides[rideNumber].rideStatus = "passengerConfirmed";
     }
   }
 }
```

Figure 4.6 code snippet ofAccept Passenger Request function.

Figure 4.6, Shows the function which allows the driver to accept a passenger's request to join a ride. It takes the ride number and the passenger's address as inputs, verifies that the caller is the driver and the ride status is "passengerRequested," and then updates the ride status to "enroute."

Reject Passenger Request: Allows the driver to reject a passenger's request to join the ride, changing the ride status back to "Initial" and refunding the driving cost to the passenger.

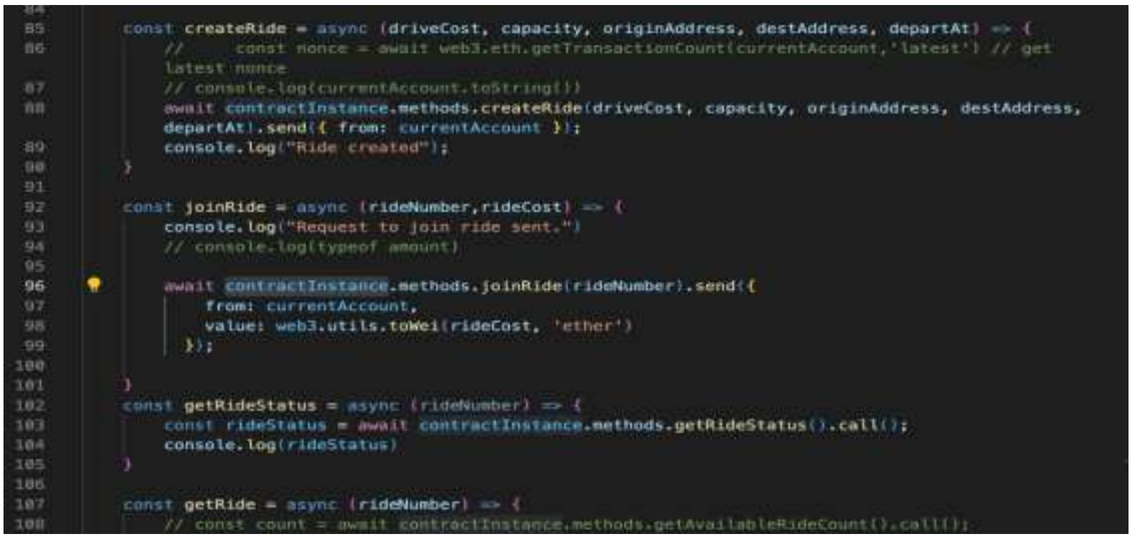


```
// reject passenger's request to ride —— called by driver
function rejectPassengerRequest(uint rideNumber, address passengerAddress) public {
Ride memory curRide = rides[rideNumber];
require(keccak256(bytes(curRide.rideStatus)) == keccak256(bytes("passengerRequested")));
require(msg.sender == curRide.driver);
for(uint i=0; i < curRide.passengerAccts.length; i++) {
   if ( curRide.passengerAccts[i] == passengerAddress) {
    rides[rideNumber].rideStatus = "Initial";
    uint256 amount = curRide.drivingCost*1000000000000000000;
    payable(curRide.passengerAccts[i]).transfer(amount);
    rides[rideNumber].passengerAccts[i] = rides[rideNumber].passengerAccts[rides[rideNumber].passengerAccts.length - 1];
    rides[rideNumber].passengerAccts.pop();        }
  }
}


}
```

Figure 4.7 code snippet of Reject Passenger Requestfunction.

Figure 4.7, Shows the function which allows the driver to reject a passenger's request to join a ride. It takes the ride number and the passenger's address as inputs, verifies that the caller is the driver and the ride status is "passengerRequested," and then removes the passenger's address from the passengerAccts array and updates the ride status to "Initial."

The front-end component of the project provides users with an interface to interact with the ride-sharing application. It includes various components, functions, and interactions with the smart contracts.



```
const createRide = async (driveCost, capacity, originAddress, destAddress, departAt) => {
    //      const nonce = await web3.eth.getTransactionCount(currentAccount,'latest') // get
    latest nonce
    // console.log(currentAccount.toString())
    await contractInstance.methods.createRide(driveCost, capacity, originAddress, destAddress,
    departAt).send({ from: currentAccount });
    console.log("Ride created");
}

const joinRide = async (rideNumber,rideCost) => {
    console.log("Request to join ride sent.")
    // console.log(typeof amount)

    await contractInstance.methods.joinRide(rideNumber).send({
        from: currentAccount,
        value: web3.utils.toWei(rideCost, 'ether')
      });

}
const getRideStatus = async (rideNumber) => {
    const rideStatus = await contractInstance.methods.getRideStatus().call();
    console.log(rideStatus)
}

const getRide = async (rideNumber) => {
    // const count = await contractInstance.methods.getAvailableRideCount().call();
```

Figure 4.8 code snippet of interact file

Figure 4.8, Shows some functions which allow the front-end to communicate with smart contracts using web3.js library. The front-end interacts with the smart contracts deployed on the Ethereum blockchain using web3.js libraries. It establishes a connection to the blockchain network, deploys contract instances, and interacts with contract functions and events. The front-end incorporates form components to capture necessary information from users. For example, the "Create Ride" form includes input fields for pickup and drop-off locations, date and time, and fare. Upon form submission, the front-endtriggers the corresponding smart contract function shown in the above figure.

VI. RESULTS

The Results section presents the outcomes and findings of the systems' implementation. This section showcases the validation of functionalities. Additionally, it provides insights into the user interface and working of the project.

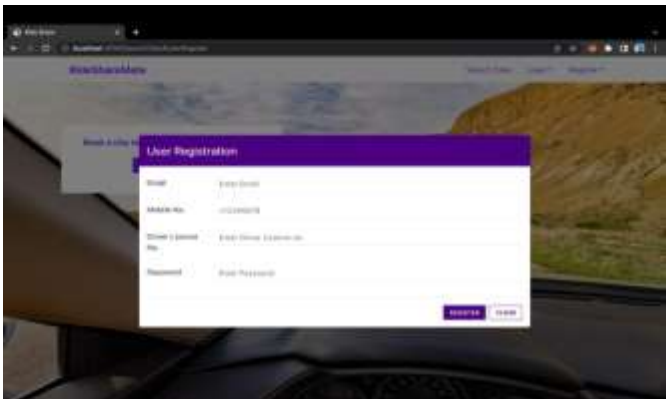


Figure 5.1 User Registration Page

Figure 5.1, Shows the registration page for the user, the registration page takes four inputs: Email, Phone, and Password and Driver License number; and the data is stored. After logging in to the app the user requires to enter the registered email address and password on the login page.

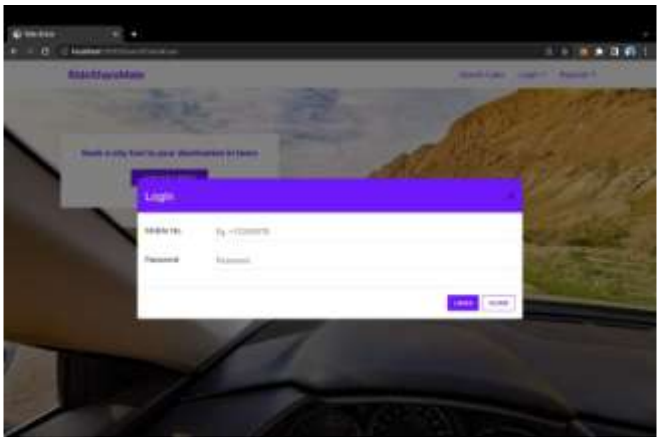


Figure 5.2 User Login Page

Figure 5.2, Shows the login page for the user, the login page takes four inputs: Phone, and Password; and the data is authenticated. After logging in to the app the user can see the user dashboard. The data stored in database is checked against the input data if the user matches they login successfully else the user needs to register his credentials first.

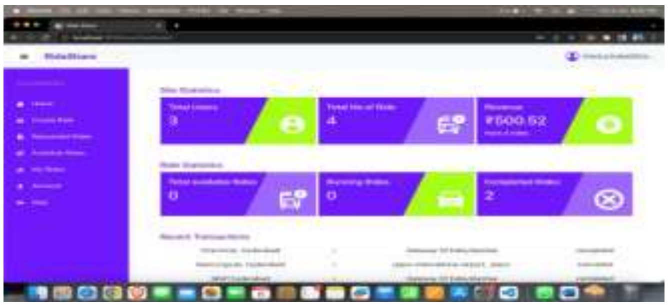


Figure 5.3 User Dashboard

Figure 5.3, Shows the dashboard of the application. The user gets a prompt to connect the meta mask wallet and the details are fetched once the wallet is connected to the application. The user dashboard has a variety of features making it transparent and convenient. The site statistics are the overall statistics of the community, and the user specific dashboard is on the left which has options to create ride, request ride, see available rides and an integrated map for convenience. At the bottom of the interface the users recent transactions are displayed which are fetched from blockchain transactions and are not stored anywhere in the database.

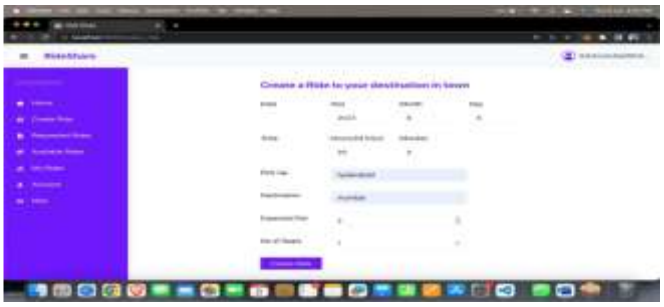


Figure 5.4 Create Ride Page

Figure 5.4, Shows the form to create a ride by a user who happens to have surplus seats and is willing to share their ride with people having a common destination.

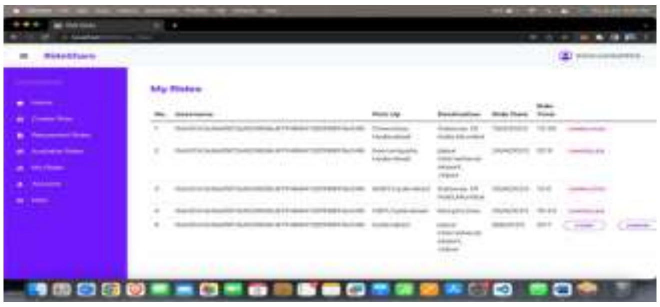


Figure 5.5 My Rides Page

Figure 5.5, Shows the Rides history of the user and their status and details like pickup, destination, ride date and ride time.

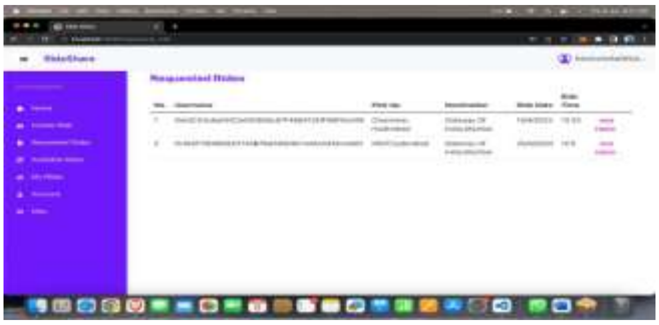


Figure 5.6 Requested Rides Page

Figure 5.6, Shows the Requested Rides of the user, their status and details like pickup, destination, ride date and ride time and status of the ride.

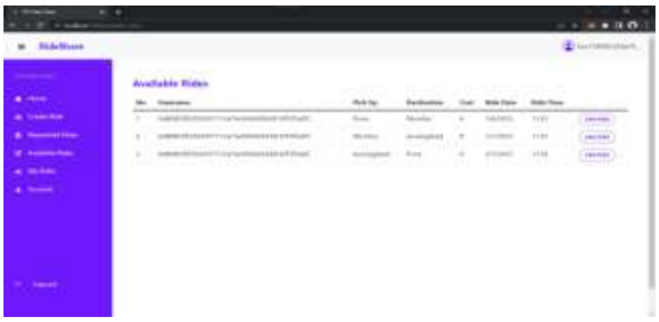


Figure 5.7 Available Rides Page

Figure 5.7, Shows the Available Rides for the user, their status and details like pickup, destination, ride date and ride time and options to join the ride, these rides are the rides active on the network.

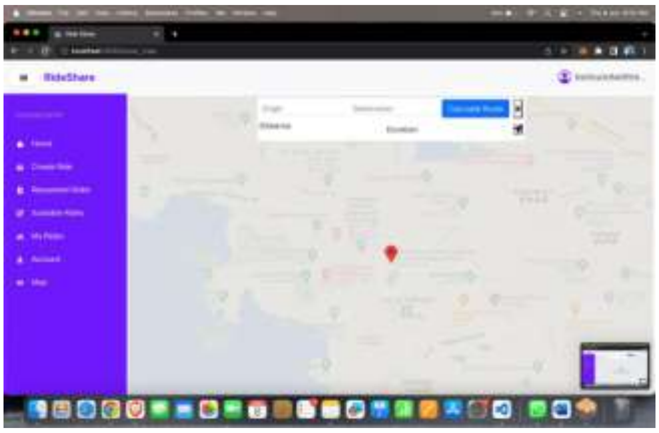
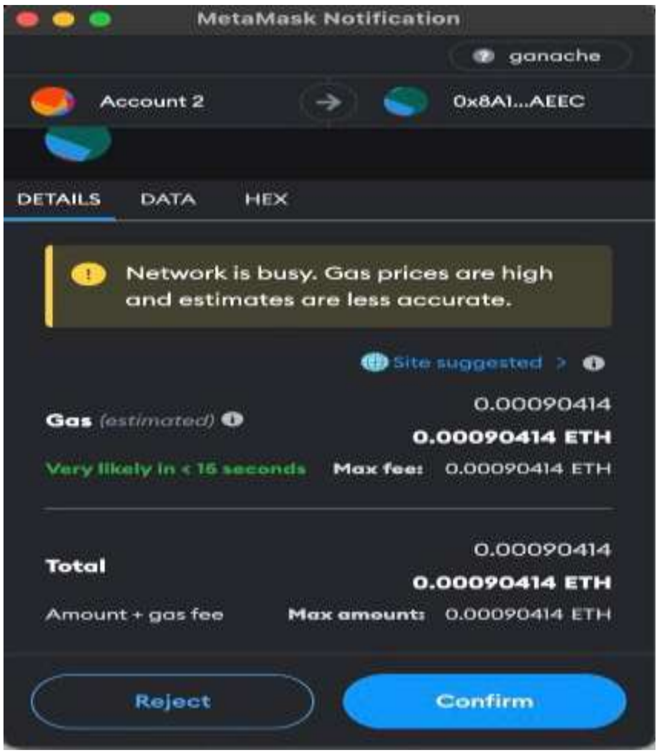


Figure 5.8 Map Page

Figure 5.8, Shows the location page, the user location, distance and duration are important factors as it is being used to create ride by both the driver and the rider. When Origin and destination are entered it calculates the distance and duration and shows it to the user.



MetaMask Notification

ganache

Account 2 → 0x8A1...AEEC

DETAILS DATA HEX

Network is busy. Gas prices are high and estimates are less accurate.

Site suggested >

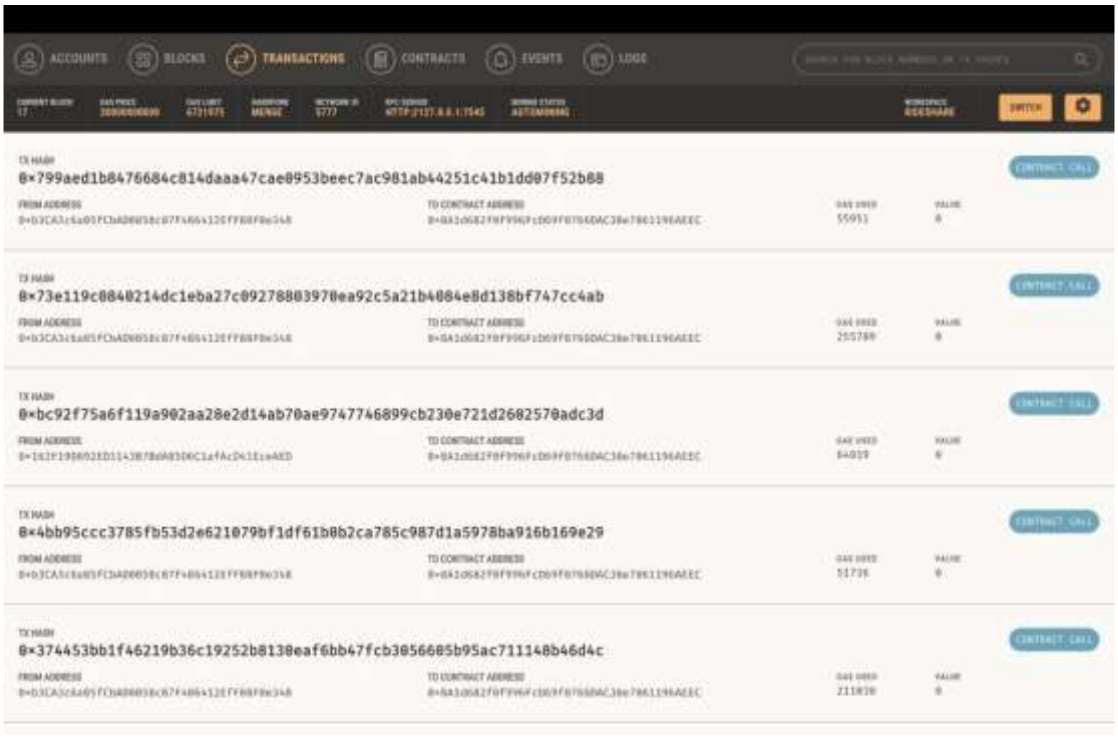
Gas (estimated) 0.00090414
0.00090414 ETH
Very likely in < 15 seconds Max fee: 0.00090414 ETH

Total 0.00090414
0.00090414 ETH
Amount + gas fee Max amount: 0.00090414 ETH

Reject Confirm

Figure 5.9 Payment Request Page on MetaMask

Figure 5.9, MetaMask transaction request page, this page displays gas fees the user has to pay.



```
TX HASH
0×799aed1b8476684c814daaa47cae0953beec7ac981ab44251c41b1dd07f52b88
FROM ADDRESS                                   TO CONTRACT ADDRESS                                 GAS USED   VALUE
0×b3CA3c6a05FCbAD0058c87F4864126FF88F0e348     0×0A1d682f0F996Fc D69F0766DAC38e7861196AEEC         55951      0

TX HASH
0×73e119c0840214dc1eba27c09278803970ea92c5a21b4084e8d138bf747cc4ab
FROM ADDRESS                                   TO CONTRACT ADDRESS                                 GAS USED   VALUE
0×b3CA3c6a05FCbAD0058c87F4864126FF88F0e348     0×0A1d682f0F996Fc D69F0766DAC38e7861196AEEC         255789     0

TX HASH
0×bc92f75a6f119a902aa28e2d14ab70ae9747746899cb230e721d2682570adc3d
FROM ADDRESS                                   TO CONTRACT ADDRESS                                 GAS USED   VALUE
0×162F190092ED1143B7BdA83D0C1afAcD41Ea4ED      0×0A1d682f0F996Fc D69F0766DAC38e7861196AEEC         84019      0

TX HASH
0×4bb95ccc3785fb53d2e621079bf1df61b0b2ca785c987d1a5978ba916b169e29
FROM ADDRESS                                   TO CONTRACT ADDRESS                                 GAS USED   VALUE
0×b3CA3c6a05FCbAD0058c87F4864126FF88F0e348     0×0A1d682f0F996Fc D69F0766DAC38e7861196AEEC         51735      0

TX HASH
0×374453bb1f46219b36c19252b8130eaf6bb47fcb3056605b95ac711148b46d4c
FROM ADDRESS                                   TO CONTRACT ADDRESS                                 GAS USED   VALUE
0×b3CA3c6a05FCbAD0058c87F4864126FF88F0e348     0×0A1d682f0F996Fc D69F0766DAC38e7861196AEEC         211830     0
```

Figure 5.10 Ganache Transactions Screen

Figure 5.10, Shows the Ganache Transactions screen, this page displays the previous transactions made by the user.

VII. CONCLUSION

This system offers a decentralized and transparent solution for the ride-sharing industry. By leveraging blockchain technology, the project addresses common challenges such as data privacy, transaction transparency, and trust issues. Through the implementation of smart contracts and distributed ledger technology, it enables direct interaction between riders and drivers, eliminating the need for centralized authorities and fostering peer-to-peer transactions. This demonstrates the potential to revolutionize the ride-sharing industry by providing improved security, increased trust, and reduced dependence on intermediaries.

VIII. FUTURE SCOPE

Looking ahead, there are several areas for future development and enhancement in the "Ride-sharing-on-Blockchain" project. Scalability and performance optimization will be crucial as the project grows and handles a larger volume of transactions. Exploring solutions like sharding, sidechains, or layer-two protocols can enhance efficiency and accommodate increased user demand. Integration with existing ride-sharing platforms presents an opportunity to leverage the project's benefits while providing a seamless experience for users and encouraging adoption.

Enhancing the user experience is a key aspect of future development. This can involve developing intuitive user interfaces, mobile applications, and user-friendly features to make the platform more accessible and Privacy and data protection also remain important considerations. Additionally, exploring opportunities to expand the project to other transportation sectors, such as freight or logistics, can unlock new possibilities and create additional value.

REFERENCES

[1] Myeonghyun Kim, Joonyoung Lee, Kisung Park, KilHoum Park, Youngho, “Design of Secure Decentralized Car-Sharing System Using Blockchain,” IEEE 2021.

[2] Binod Vaidya, Hussein T.Mouftah, “Security for Shared Electric and Automated Mobility Services Mobility Services in Smart Cities,” IEEE 2020.

[3] Sophia auer, Somanath mazumdar, Raghavendra roa, “Towards blockchain-IOT based shared mobility: Car-sharing and leasing as a case study,” Journal of Network and Computer Applications 2022.

[4] Ali Dorri, Marco Steger, Salil S. Kanhere, Raja Jurdak, “A distributed Solution to

Automotive Security and Privacy," IEEE 2017.

[5] Jiao Jiao, Shuanglong Kan, Shang-wei Lin, David Sanan, Yang Liu, Jun Sun, "Semantic Understanding of Smart Contracts: Executable Operational Semantics of Solidity," IEEE 2020.

[6] Baza, M. Mahmoud, G. Srivastava, W. Alasmary, M. Younis, "A light blockchain-powered privacy-preserving organization scheme for ride-sharing services," IEEE, vol. 10, pp. 9–19, (2021).

[7] W. Li, C. Meese, H. Guo, M. Nejad, "Blockchain-enabled identity verification for safe ridesharing leveraging zeroknowledge proof," IEEE, vol. 11, pp. 41–63, (2020).

[8] P. Pal, S. Ruj, "BlockV: A blockchain-enabled peer-peer ride-sharing service," IEEE, vol. 13, pp. 142–167,(2019).

[9] M. Li, L. Zhu, and X. Lin, "Efficient and PrivacyPreserving Carpooling Using Blockchain-Assisted Vehicular Fog Computing," IEEE, vol. 6, pp. 4573 - 4584, (2019).