VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590018

Fifth Semester Project Report On

"COMMUTEBUDDY-ETHEREUM BASED CARPOOLING DAPP"

Submitted in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering

in

Department of CSE(ICB)

Submitted by

1BI22IC017	DHANYA N MOGER
1BI22IC026	INDU C N
1BI22IC028	JYOTHIKA SUDARSAN
1BI22IC034	L TEJASWINI REDDY

Under the guidance of

Prof. SWETHA M
Assistant Professor
Department of MCA

BANGALORE INSTITUTE OF TECHNOLOGY

K R Road, V V Pura, Bengaluru-560004

Affiliated to VTU, Belagavi, Approved by AICTE, Accredited by NBA, NAAC

2024-2025

BANGALORE INSTITUTE OF TECHNOLOGY

Department of CSE-ICB

K R Road, V V Pura, Bengaluru-560004.

CERTIFICATE

This is to certify that the Fifth Semester Mini project entitled **"COMMUTEBUDDY-ETHEREUM BASED CARPOOLING DAPP"** is carried out by **DHANYA N MOGER(1BI22IC017), INDU C.N (1BI22IC026), JYOTHIKA SUDARSAN (1BI22IC028), L TEJASWINI REDDY (1BI22IC034)**, bonafide students of Bangalore Institute of Technology, Bengaluru, in partial fulfilment of the requirements for the award of Bachelor of Engineering in Computer Science (Iot,CyberSecurity with Blockchain Technology). This fulfils all the requirements of the regulations for the award of the degree. The contents of this report have not been submitted to any other institute or university for the award of any degree or diploma and are not a repetition of the work carried out by others.

Prof. SWETHA M
Project Guide & Assistant Professor
Dept.OF MCA
BIT, Bengaluru

Dr. Shivakumar B R
Associate Professor & HOD
Dept.CSE(ICB)
BIT, Bengaluru

Dr. Aswath M U
Principal
BIT, Bengaluru

Project Final Viva Voce Examination

Examiners	Signature with Date
Examiner 1	
Examiner 2	

BANGALORE INSTITUTE OF TECHNOLOGY

Department of CSE-ICB

K R Road, V V Pura, Bengaluru-560004.

DECLARATION

We hereby declare that the work embodied in this project report titled **"COMMUTEBUDDY-ETHEREUM BASED CARPOOLING DAPP"** is the result of original work carried out by **DHANYA N MOGER (1BI22IC017)**, **INDU C N (1BI22IC026)**, **JYOTHIKA SUDARSAN(1BI22IC028)** and **L TEJASWINI REDDY(1BI22IC034)** at Bangalore Institute of Technology, Department of Computer Science (Iot,CyberSecurity with Blockchain Technology), Bengaluru-560004, under the guidance of **Prof. SWETHA M**.This report has not been submitted in part or full for the award of any diploma or degree of this or any other university.

USN	Name	Student Signature with Date
1BI22IC017	DHANYA N MOGER	
1BI22IC026	INDU C N	
1BI22IC028	JYOTHIKA SUDARSAN	
1BI22IC034	L TEJASWINI REDDY	

ACKNOWLEDGEMENT

First and foremost, We would like to extend my deepest gratitude to my project guide, **Prof. SWETHA M**, Assistant Professor, Department of MCA, BIT, Bengaluru. His/Her unwavering technical expertise, insightful guidance, and moral encouragement have been pivotal in navigating the challenges and achieving the objectives of this project. His mentorship has left an indelible mark on our academic and personal growth.

We are profoundly grateful to **Dr. Aswath M.U**, Principal of BIT, for fostering an environment of academic excellence and innovation.We also extend my sincere appreciation to **Dr. Shivakumar B R**, Associate Professor and HOD of the Department of CSE(ICB), for his/her leadership and constant encouragement throughout the course of this work. Our heartfelt thanks go to the faculty and staff of the Department of CSE(ICB), whose support and resources have played a crucial role in facilitating this project.

Special acknowledgment to our Project Coordinator **Prof. SWETHA M**, Assistant Professor, Department of MCA, Bengaluru for her invaluable advice and consistent support that helped shape this project at various stages.

On a personal note, We owe our heartfelt gratitude to our parents for their unconditional love, sacrifices, and unwavering belief in our capabilities.Their encouragement has been the cornerstone for our perseverance and success.

Lastly, We extend our sincere thanks to our friends, peers, and all those who contributed directly or indirectly to the successful completion of this project. Your camaraderie, insights, and encouragement have been a source of motivation throughout this journey.

1BI22IC017	DHANYA N MOGER
1BI22IC026	INDU C N
1BI22IC028	JYOTHIKA SUDARSAN
1BI22IC034	L TEJASWINI REDDY

Abstract

Carpooling, a practical solution to reduce traffic congestion and environmental impact, faces significant challenges in its traditional implementation.Current systems, often managed by centralized platforms, suffer from issues such as a lack of transparency, difficulty in resolving disputes, and limited user trust.Additionally, centralized systems are vulnerable to data breaches and single points of failure, undermining user confidence and limiting adoption.These inefficiencies highlight the need for a more secure, transparent, and reliable approach to carpooling. Blockchain technology offers a promising solution by decentralizing the management of carpooling systems. Leveraging Ethereum, a leading blockchain platform, enables the development of a decentralized application (dApp) that ensures secure and tamper-proof transactions. By using smart contracts—self-executing code stored on the blockchain—users can interact in a trustless environment, eliminating the need for intermediaries and ensuring automated enforcement of rules.The proposed Ethereum-based carpooling dApp incorporates several cutting-edge tools and technologies to achieve this vision.Solidity, Ethereum's native programming language, is used to develop smart contracts that manage core functionalities such as ride booking, payment processing, and user verification.The dApp utilizes Web3.js, a JavaScript library, to bridge the frontend and blockchain backend, enabling seamless interaction between users and the Ethereum network.Tools such as Remix are employed for writing and deploying smart contracts, while Ganache simulates a local blockchain environment for testing and debugging.Concurrently, the databank smart contract enables decentralized data storage and retrievalS.

Contents

Certificate i

Declaration iii

Acknowledgements iv

Abstract v

List of Figures ix

List of Tables x

Abbreviations xi

1 Introduction 1
- 1.1 Objectives . . . 1
- 1.2 Existing Systems . . . 2
- 1.3 Motivation . . . 2

2 Literature Survey 4
- 2.1 Drawbacks of Existing Sysytems . . . 4
- 2.2 Problem Statement . . . 5
- 2.3 Proposed Solution . . . 5

3 Problem Statement 8

4 Project Planning 10

5 System Design 14
- 5.1 High-Level System Architecture . . . 14
  - 5.1.1 Frontend Layer (User Interface) . . . 14
  - 5.1.2 Backend Layer . . . 15
  - 5.1.3 Blockchain Layer (Ethereum Network) . . . 15
  - 5.1.4 Wallet Integration (MetaMask) . . . 16
- 5.2 System Components and Data Flow . . . 16
  - 5.2.1 User Registration and Authentication . . . 16
  - 5.2.2 Ride Creation (by Drivers) . . . 16
  - 5.2.3 Ride Search and Booking (by Riders) . . . 17
  - 5.2.4 Payment Handling . . . 17
  - 5.2.5 Rating System . . . 17
  - 5.2.6 Emergency Response . . . 18

5.3 Database Design (for Off-chain Data) . . . 18
5.4 Technology Stack . . . 18
5.5 Security Considerations . . . 19
5.6 System Workflow Diagram . . . 19

6 Software Requirements Specifications 20

7 Modelling 23
7.1 Analytical Modeling for Ethereum-based Carpooling DApp . . . 23
7.1.1 Smart Contract Model . . . 23
7.1.2 Preprocessing and Data Handling . . . 24
7.1.3 Evaluation Metrics and Performance . . . 24
7.2 Odd Paradigm . . . 25
7.2.1 Use Case Diagram . . . 26
7.2.2 Sequence Diagram . . . 28

8 Design 31
8.1 High-LevelDesign(HLD): System Architecture . . . 31
8.1.1 System Architecture Overview . . . 31
8.1.2 Components and Communication Flow . . . 31
8.1.3 Interaction Flow in the System . . . 33
8.2 Detailed Design . . . 34
8.2.1 Detailed Design Components . . . 34
8.2.2 Database Design . . . 37
8.2.3 Interaction Flow (Detailed) . . . 39

9 Implementation with Source Code 41
9.1 HARDWARE ARCHITECTURE . . . 41
9.1.1 Components and Their Roles . . . 41
9.2 SOFTWARE AND DESIGN . . . 42
9.2.1 Programming Languages and Tools . . . 42
9.2.2 Ride Booking System . . . 42
9.2.3 Payment Handling . . . 43
9.2.4 User Rating System . . . 43
9.2.5 Emergency Module . . . 43
9.3 SYSTEM WORKFLOW . . . 44
9.3.1 Initialization . . . 44
9.3.2 Ride Creation . . . 44
9.3.3 Ride Search and Booking . . . 44
9.3.4 Ride Completion . . . 44
9.3.5 User Rating . . . 44
9.4 SYSTEM OVERVIEW . . . 44
9.5 CODE SNIPPETS . . . 45

10 Results 48

11 Conclusion 52
11.1 Development of Ethereum-Based Carpooling dApp . . . 52
11.2 Future Scope . . . 53

11.3 Applications 53

List of Figures

7.1 Data Flow Diagram 25
7.2 Usecase Diagram of Ehereum based carpooling dApp 28
7.3 Sequence Diagram of Ehereum based carpooling dApp 29

8.1 Driver and Passanger login Panel 35
8.2 Menu Option Dashboard 37
8.3 Ride Confirmation 39

9.1 Ganache Software 42
9.2 Mapping Data 45
9.3 User Registration 46
9.4 ABI Code 47
9.5 User Registration 47

List of Tables

Nomenclature

AI	Artificial Intelligence
IoT	Internet of Things
WSN	Wireless Network Networks

Chapter 1

Introduction

An Ethereum-based Carpooling DApp (Decentralized Application) leverages blockchain technology to create a decentralized platform for ride-sharing services.It enables users to efficiently match riders with drivers while ensuring transparency, security, and trust through smart contracts.The DApp operates on the Ethereum blockchain, allowing for secure payment processing and eliminating the need for intermediaries.Riders and drivers can directly interact, ensuring a seamless experience, while smart contracts automate ride matching and payment execution.This decentralized approach enhances user privacy, reduces costs, and fosters a more efficient and transparent carpooling system.

1.1 Objectives

The primary objective is to design and build a decentralized platform that redefines the carpooling experience through the use of Ethereum-based smart contracts.This platform aims to eliminate reliance on centralized intermediaries, ensuring that all transactions and processes are executed transparently and securely on the blockchain.The system will be user-friendly, with an intuitive interface that simplifies the onboarding process for users with varying levels of technical expertise.It will incorporate features such as secure user authentication, real-time ride matching, automated fare calculation, and seamless payment processing, all powered by smart contracts.By decentralizing operations, the platform ensures that users have full visibility into the system, fostering trust and eliminating the risk of manipulation or fraud.The use of smart contracts also guarantees that terms and

conditions are enforced automatically, reducing the potential for disputes and enhancing operational efficiency.Moreover, the platform will prioritize scalability and adaptability, allowing for the inclusion of additional features or integration with other decentralized applications in the future.By bridging the gap between blockchain technology and practical use cases, the platform will not only make carpooling more efficient but also set a standard for decentralized solutions in mobility.

1.2 Existing Systems

The existing carpooling systems primarily rely on centralized platforms, which act as intermediaries connecting ride seekers and providers.While these systems offer convenience and ease of use, they are plagued by several limitations.Users face a lack of transparency regarding how fares are calculated, how their data is managed, and how disputes are resolved.Additionally, centralized data storage raises significant privacy concerns, making personal and financial information vulnerable to breaches.High transaction fees imposed by intermediaries further increase the cost of using these services.Furthermore, the reliance on a single point of failure, such as a central server, makes the system susceptible to outages and cyberattacks.These issues, combined with limited scalability and an absence of mechanisms to promote sustainable practices, highlight the inefficiencies of the current systems and underscore the need for a decentralized, transparent, and secure alternative.

1.3 Motivation

The motivation behind developing an Ethereum-based carpooling decentralized application (dApp) stems from a pressing need to address the significant limitations and inefficiencies of traditional carpooling systems.Centralized platforms often suffer from a lack of transparency, leading to mistrust among users due to limited visibility into how their transactions are processed and how their data is managed.Additionally, these systems are vulnerable to data breaches, which compromise sensitive user information and expose centralized databases to hacking

and unauthorized access.This vulnerability undermines user confidence and raises serious security concerns.Moreover, handling disputes in traditional carpooling systems can be cumbersome and biased, with users frequently encountering difficulties when resolving issues related to ride cancellations, fare discrepancies, or service quality.The high service fees charged by centralized platforms further reduce the cost-effectiveness of carpooling for users, making it a less attractive option.Consequently, the overall adoption of carpooling services remains limited due to these persistent challenges. Blockchain technology offers a transformative solution to these issues by decentralizing the management of carpooling systems.By eliminating intermediaries, blockchain enables direct peer-to-peer transactions between riders and drivers, significantly reducing operational costs and enhancing service efficiency.The inherent transparency of blockchain technology ensures that all transactions are immutably recorded, providing a tamper-proof history of rides, payments, and user ratings.This transparency builds trust among users and ensures accountability.Additionally, blockchain's decentralized nature and cryptographic principles offer robust security for user data, making unauthorized alterations virtually impossible.Smart contracts, a core feature of blockchain technology, automate key processes such as ride bookings, payment settlements, and dispute resolutions, ensuring fair and reliable operations without the need for manual intervention.Users also gain greater control over their data and transactions, enhancing privacy and reducing the risk of data misuse.

Chapter 2

Literature Survey

2.1 Drawbacks of Existing Sysytems

Existing carpooling systems, typically managed by centralized platforms, face several significant drawbacks that hinder their effectiveness and user adoption. These drawbacks include:

1. Lack of Transparency: Centralized carpooling services often operate without sufficient transparency, leading to trust issues among users.Riders and drivers have limited visibility into how their transactions are processed and how their data is managed.

2. Data Security Concerns: Centralized databases are prone to data breaches, risking the exposure of sensitive user information.The reliance on a single point of control makes these systems vulnerable to hacking and unauthorized access.

3. Dispute Resolution: Handling disputes in centralized systems can be cumbersome and biased.Users frequently encounter difficulties when resolving issues related to ride cancellations, fare discrepancies, or service quality.The lack of an efficient and impartial dispute resolution mechanism can lead to user frustration and dissatisfaction.

4. High Service Fees: Centralized platforms typically charge substantial service fees to cover operational costs.These fees reduce the cost-effectiveness of carpooling for users, making it a less attractive option.High fees can discourage potential users from adopting carpooling services.

5. Limited User Trust: Due to the aforementioned issues, users may lack confidence

in the fairness and security of centralized carpooling services.This lack of trust limits widespread adoption and hinders the growth of carpooling as a viable transportation solution.

2.2 Problem Statement

Traditional carpooling systems, typically managed by centralized platforms, face several critical issues that limit their effectiveness and adoption.These problems include:

1. Lack of Transparency: Users have limited visibility into transaction processes and data management, leading to mistrust.

2. Data Security Concerns: Centralized databases are vulnerable to data breaches, compromising user information and undermining confidence in the system.

3. Inefficient Dispute Resolution: Dispute handling can be biased and cumbersome, causing frustration over ride cancellations, fare discrepancies, or service quality issues.

4. High Service Fees: Centralized platforms charge substantial fees, reducing the cost-effectiveness of carpooling and deterring potential users.

5. Limited User Trust: The aforementioned issues collectively reduce user trust in centralized carpooling services, limiting widespread adoption.

6. Single Point of Failure: Centralized systems are prone to failures that can disrupt the entire service, negatively impacting reliability.

7. Privacy Concerns: Users have minimal control over their personal data, raising concerns about potential misuse by centralized entities.

8. Operational Inefficiencies: Delays in processing transactions and matching riders with drivers lead to longer wait times and unsatisfactory user experiences.

2.3 Proposed Solution

To address these challenges, the proposed solution is to develop an Ethereum-based carpooling decentralized application (dApp). The proposed solutions include:

1. Decentralization

- **Solution:**Utilize Ethereum blockchain to eliminate intermediaries, enabling direct peer-to-peer transactions between riders and drivers.
- **Benefit:** Reduces operational costs and enhances service efficiency.

2.Transparency:

- **Solution:**Implement immutable transaction records on the blockchain.
- **Benefit:** Provides a transparent and tamper-proof history of rides, payments, and user ratings, building trust among users.

3.Data Security:

- **Solution:**Use blockchain's cryptographic principles to secure user data and transactions..
- **Benefit:**Prevents unauthorized access and ensures data integrity.

4.Automated Dispute Resolution:

- **Solution:**Develop smart contracts to automate ride bookings, payments, and dispute resolutions.
- **Benefit:**Ensures fair and reliable operations without manual intervention, reducing user frustration.

5. Reduced Service Fees:

- **Solution:**Minimize fees by eliminating centralized intermediaries.
- **Benefit:**Enhances the cost-effectiveness of carpooling, making it more attractive to users.

6. Enhanced User Trust:

- **Solution:**Leverage the transparency and security features of blockchain.
- **Benefit:**Increases user confidence in the system, promoting wider adoption.

7. Resilience and Reliability

- **Solution:**Deploy the dApp on a decentralized network to avoid single points of failure.
- **Benefit:**Ensures continuous service availability and reliability.

8. User Privacy:

- **Solution:**Implement decentralized identity solutions, allowing users to control their personal data.
- **Benefit:**Enhances privacy and reduces the risk of data misuse.

9. Operational Efficiency:

- **Solution:**Use blockchain for efficient transaction processing and ride matching.
- **Benefit:**Reduces wait times and improves user experience

Chapter 3

Problem Statement

The traditional carpooling systems, typically managed by centralized platforms, face several critical issues that hinder their effectiveness, limit user adoption, and reduce overall satisfaction.These issues can be detailed as follows:

1. Lack of Transparency

Centralized carpooling services often operate without sufficient transparency.Users, both riders and drivers, have limited visibility into the inner workings of these platforms, such as how transactions are processed, how fares are calculated, and how disputes are resolved.For instance, riders may feel uncertain about whether the fare they are paying is fair, and drivers may be unsure if they are receiving their due payments.

2. Data Security Concerns

Centralized carpooling platforms store vast amounts of sensitive user data, including personal details, ride histories, and payment information.These centralized databases are attractive targets for cyber-attacks and data breaches.A successful breach can lead to the exposure of sensitive user information, causing significant harm to users and damaging the platform's reputation.

3. Inefficient Dispute Resolution

Disputes are common in carpooling scenarios, whether they pertain to fare disagreements, ride cancellations, or service quality issues.Centralized platforms often struggle with handling these disputes efficiently and impartially.The resolution

process can be slow, biased, and frustrating for users.

4. High Service Fees

To sustain their operations, centralized carpooling platforms charge service fees on each transaction.These fees can be substantial, eating into the savings that riders and drivers hope to achieve through carpooling.High fees reduce the overall cost-effectiveness of carpooling, making it less attractive compared to other transportation options.

5. Limited User Trust

The combination of the aforementioned issues—lack of transparency, data security concerns, inefficient dispute resolution, and high service fees—results in limited user trust.Users may be hesitant to use centralized carpooling services due to fears of unfair treatment, data breaches, and hidden costs.This lack of trust is a significant barrier to widespread adoption of carpooling services.

6. Single Point of Failure

Centralized systems are inherently vulnerable to single points of failure.If the central server or the platform experiences downtime, technical issues, or cyber-attacks, the entire service can be disrupted.Such disruptions can lead to significant inconvenience for users, who rely on the platform for their daily commutes.This lack of resilience undermines the reliability of the service.

7. Privacy Concerns

In centralized carpooling platforms, users have minimal control over their personal data.The platform has access to and control over user data, which can be used for various purposes, sometimes without the explicit consent of users.This situation raises privacy concerns, as users may be uncomfortable with the potential misuse of their personal information.

Chapter 4

Project Planning

Effective project planning is crucial to ensure the successful development and deployment of the Ethereum-based carpooling decentralized application.This planning involves several phases, each focusing on different aspects of the project, from initial conception to final deployment and maintenance.

1.Project Intiation

1. **Objectives:**
   - Define the project scope and goals.
   - Identify stakeholders and their roles.
   - Establish project requirements and deliverables.
2. **Key Activities:**
   - Conduct a project kickoff meeting.
   - Define the project scope, objectives, and deliverables.
   - Identify key stakeholders and form a project team.
   - Develop a project charter and obtain approval from stakeholders.

2. Requirement Analysis

1. **Objectives:**
   - Gather detailed requirements from stakeholders.
   - Define functional and non-functional requirements.
   - Develop use case diagrams and user stories.

2. **Key Activities:**
   - Conduct requirement gathering sessions with stakeholders.
   - Document functional requirements (e.g., ride booking, payment processing).
   - Document non-functional requirements (e.g., security, performance).
   - Develop use case diagrams and user stories.
   - Obtain stakeholder approval for the requirements document.

3. System Design

1. **Objectives:**
   - Design the overall system architecture.
   - Define the database schema and smart contract interfaces.
   - Plan the frontend and backend components

2. **Key Activities:**
   - Develop system architecture diagrams.
   - Design the database schema.
   - Define the smart contract interfaces and functions.
   - Plan the frontend architecture and user interface design.
   - Create a detailed design document and get approval from stakeholders.

4. Development Setup

1. **Objectives:**
   - Set up the development environment.
   - Install necessary tools and frameworks.
   - Establish version control and project management tools

2. **Key Activities:**
   - Set up a development environment with necessary software (e.g., VS Code, Remix IDE, Ganache).
   - Install required libraries and frameworks (e.g., Web3.js, Solidity).
   - Configure version control (e.g., Git) and project management tools (e.g., Jira, Trello).

5. Smart Contract Development

1. **Objectives:**
   - Develop and test smart contracts for the carpooling dApp.
   - Ensure smart contracts are secure and efficient.
2. **Key Activities:**
   - Write smart contracts in Solidity for key functionalities (e.g., ride booking, payment processing, user verification)
   - Test smart contracts using tools like Ganache for local blockchain simulation.
   - Conduct security audits and optimize smart contracts for gas efficiency.
   - Deploy smart contracts to an Ethereum test network (e.g., Rinkeby, Goerli) for further testing.

6. Frontend and Backend Development

1. **Objectives:**
   - Develop the frontend and backend components of the dApp.
   - Integrate frontend with blockchain through Web3.js.
2. **Key Activities:**
   - Develop the user interface using HTML, CSS, JavaScript, and frameworks like React.
   - Implement backend services using Node.js to handle communication between the frontend and blockchain.
   - Integrate the frontend with smart contracts using Web3.js.
   - Ensure proper authentication and wallet integration using MetaMask.

7. Testing and Quality Assurance

1. **Objectives:**
   - Ensure the dApp functions as intended and is free of bugs.
   - Validate the performance, security, and usability of the dApp.
2. **Key Activities:**
   - Write and execute test cases for smart contracts using frameworks like

Mocha and Chai.

• Perform unit testing, integration testing, and end-to-end testing for the frontend and backend.

• Conduct user acceptance testing (UAT) with stakeholders.

• Perform security testing and audits to identify and fix vulnerabilities.

• Optimize performance to ensure a smooth user experience.

8. Deployment

1. **Objectives:**
   - Deploy the dApp to the Ethereum mainnet.
   - Ensure all components are properly configured and functional
2. **Key Activities:**
   - Deploy the smart contracts to the Ethereum mainnet.
   - Configure the frontend to interact with the mainnet smart contracts.
   - Perform final testing to ensure everything is working correctly.
   - Launch the dApp and conduct a post-launch review with stakeholders.

9. Maintenance and Updates

1. **Objectives:**
   - Provide ongoing support and maintenance for the dApp.
   - Implement updates and new features based on user feedback.
2. **Key Activities:**
   - Monitor the dApp for any issues or bugs and address them promptly.
   - Gather user feedback and identify areas for improvement.
   - Plan and implement updates and new features.
   - Ensure the dApp remains secure and compliant with any regulatory changes.

By following this detailed project plan, the development team can ensure the successful creation and deployment of the Ethereum-based carpooling dApp, addressing the limitations of traditional systems and providing a secure, transparent, and efficient platform for users.

Chapter 5

System Design

The system design for the Ethereum-based carpooling decentralized application (DApp) encompasses both the high-level architecture and the detailed components required to implement a decentralized ride-sharing platform.The system design ensures seamless interactions between users, smart contracts, the Ethereum blockchain, and the frontend/backend components of the DApp.

5.1 High-Level System Architecture

The Ethereum-based carpooling DApp follows a decentralized architecture where users interact directly with the blockchain.The system can be broken down into several key components:

5.1.1 Frontend Layer (User Interface)

- **Technology:** HTML, CSS, JavaScript, React.js
- **Functionality:**
    1. Provides the user interface for both riders and drivers to interact with the DApp.
    2. Allows users to sign up, log in, search for rides, book rides, and view ride details.
    3. Integrates with the Ethereum blockchain through Web3.js to interact with smart contracts.

4. Wallet integration with MetaMask for user authentication and transaction signing.

5.1.2 Backend Layer

- **Technology:** Node.js, Express.js
- **Functionality:**
  1. Acts as an intermediary between the frontend and blockchain.
  2. Handles user requests and interacts with the Ethereum blockchain through Web3.js.
  3. Provides additional services like user authentication, profile management, and ride creation.
  4. Stores non-sensitive data in a database (e.g., MongoDB) if needed (e.g., ride history, user preferences).
  5. Handles error management, transaction events, and off-chain data.

5.1.3 Blockchain Layer (Ethereum Network)

- **Technology:** Ethereum, Solidity, Web3.js
- **Functionality:**
  1. The Ethereum blockchain stores the DApp's data and ensures transparency, security, and immutability.
  2. Smart Contracts (written in Solidity) handle core functionalities such as:
     (a) Ride Creation: Allows drivers to list new rides with details such as source, destination, price, and available seats.
     (b) Ride Booking: Allows riders to book seats, with smart contracts ensuring proper payment handling and seat allocation.
     (c) Payment Processing: Handles the transfer of payments to drivers only after the ride has been completed successfully, using smart contracts to ensure fair transactions.

(d) Dispute Resolution: Automates dispute resolution through the use of predefined rules in the smart contracts.

(e) User Ratings: Collects ratings for both riders and drivers, which are stored immutably on the blockchain.

5.1.4 Wallet Integration (MetaMask)

- **Technology:** MetaMask (Browser Extension)
- **Functionality:**

1. Users connect their wallets (such as MetaMask) to the DApp to authenticate their identity and interact with the Ethereum blockchain.
2. Allows users to approve transactions (ride booking, payments) and sign interactions with smart contracts.

5.2 System Components and Data Flow

5.2.1 User Registration and Authentication

- **Flow:**

1. Users (riders or drivers) register by connecting their MetaMask wallet to the DApp.
2. Their Ethereum address is used as a unique identifier.
3. A simple profile is created (username, preferences) and stored on-chain using the Ethereum network or off-chain in a decentralized storage system (e.g., IPFS).

5.2.2 Ride Creation (by Drivers)

- **Flow:**

1. Drivers create a new ride by entering details such as source, destination, price, number of available seats, etc.

2. This information is stored in the smart contract and added to the blockchain.
3. The ride data is immutable and can be viewed by riders through the DApp.

5.2.3 Ride Search and Booking (by Riders)

- **Flow:**

1. Riders search for available rides by entering their source and destination locations.
2. The DApp interacts with the Ethereum smart contract to fetch available rides and display them.
3. Riders can book a ride by selecting one and confirming their seat, with payment processed via the integrated MetaMask wallet.
4. The payment is held in escrow until the ride is completed, ensuring security for both parties.

5.2.4 Payment Handling

- **Flow:**

1. The rider's payment is processed through MetaMask and stored in the smart contract.
2. Payment is only released to the driver after the ride is marked as completed by the driver, ensuring that the driver is paid fairly for the ride.
3. If any issues arise, the smart contract includes a mechanism for dispute resolution, allowing users to resolve conflicts based on predefined rules.

5.2.5 Rating System

- **Flow:**

1. After completing a ride, both the rider and driver rate each other using a 5-star system.

2. The ratings are submitted to the smart contract, which stores them immutably on the blockchain.
3. This helps build trust and accountability within the community.

5.2.6 Emergency Response

- **Flow:**

1. In case of emergencies during the ride, the rider can trigger an emergency response feature.
2. The rider's GPS location is sent to predefined emergency contacts via a smart contract, ensuring timely help if needed.

5.3 Database Design (for Off-chain Data)

While the Ethereum blockchain will store crucial data like transactions, ride details, and user ratings, the system may also store non-sensitive data off-chain, such as:

- **Users Table:** Stores user profile information (e.g., username, preferences).
- **Ride Listings Table:** Contains details of rides created by drivers (e.g., source, destination, price, availability).
- **Ride History Table:** Keeps track of completed rides, payments, and user interactions.

This data can be stored in a database like MongoDB or any decentralized storage like IPFS if desired.

5.4 Technology Stack

- **Frontend:** HTML, CSS, JavaScript, React.js, Web3.js (for blockchain integration).
- **Backend:** Node.js, Express.js (for handling backend logic and APIs).
- **Smart Contracts:** Solidity, Ethereum.

- **Blockchain:** Ethereum (test networks like Rinkeby, Goerli, and mainnet for deployment).
- **Wallet Integration:** MetaMask (for transaction signing and user authentication).
- **Local Blockchain Simulation:** Ganache (for development and testing).
- **Database:** MongoDB (for off-chain data storage).

5.5 Security Considerations

- **Smart Contract Audits:** Before deployment, the smart contracts will undergo a security audit to ensure there are no vulnerabilities.
- **Wallet Encryption:** The MetaMask wallet ensures private keys are stored securely on the user's device, reducing the risk of key theft.
- **HTTPS:** The DApp frontend will use HTTPS to ensure secure communication between the client and server.
- **Data Encryption:** Any sensitive off-chain data stored in the database will be encrypted to protect users' privacy.

5.6 System Workflow Diagram

- User Registration → MetaMask Wallet Authentication.
- Driver Creates Ride → Store Ride Data on Blockchain via Smart Contract.
- Rider Searches for Ride → Fetch Ride Data from Blockchain.
- Rider Books Ride → Transaction Processed via Smart Contract.
- Ride Completed → Payment Released to Driver.
- User Rates Ride → Store Ratings on Blockchain.
- Emergency Trigger → Send Real-time GPS Location via Smart Contract.

Chapter 6

Software Requirements Specifications

This document outlines the requirements for the Ethereum-based carpooling dApp, which will enable decentralized ride-sharing, ensuring transparency, security, and efficiency.

Scope

The dApp allows drivers to create rides, riders to book seats, secure payment processing via Ethereum smart contracts, automated dispute resolution, and user ratings. It also includes an emergency alert system for riders.

Definitions and Abbreviations

- **dApp:** Decentralized Application.
- **Ethereum:** Blockchain platform for smart contracts.
- **MetaMask:** Ethereum wallet for transaction management.
- **Web3.js:** JavaScript library for Ethereum blockchain interaction.
- **Ganache:** Local blockchain for testing.

Product Features

- **User Registration:** Authentication using MetaMask.

- **Ride Creation:** Drivers can create rides with details such as source, destination, price, and available seats.
- **Ride Search/Booking:** Riders can search for rides by source and destination and book available seats.
- **Payment Handling:** Payment is held in escrow via smart contracts and released upon ride completion.
- **User Ratings:**Both riders and drivers can rate each other, with ratings stored immutably on the blockchain.
- **Emergency Response:**Riders can trigger an emergency alert with GPS coordinates sent to predefined contacts.

User Characteristics

- **Drivers:**Create and manage rides.
- **Riders:**Search for and book rides.

System Features Functional Requirements

The system encompasses several essential features and functional requirements.For user registration and authentication, users authenticate through MetaMask, utilizing their Ethereum address as a unique identifier.Drivers have the capability to create a ride by providing pertinent details such as source, destination, available seats, and price, with ride details securely stored on the Ethereum blockchain.Riders can search for available rides based on their desired source and destination, book a seat, and complete payment through MetaMask.

Non-Functional Requirements

Performance: The system should support up to 1,000 concurrent users and complete transactions within 5-10 seconds.

Security: The system should use MetaMask for secure wallet integration and smart contracts must undergo security audits.

Usability: The user interface should be intuitive and mobile-responsive to provide a smooth experience for both riders and drivers.

Availability: The dApp should be accessible 24/7, with minimal downtime for maintenance.

External Interface Requirements

Hardware Interfaces

Users must have an internet-enabled device (smartphone, tablet, laptop) to interact with the dApp.

Software Interfaces

- MetaMask: Used for wallet management and authentication.
- Web3.js: Used to interact with the Ethereum blockchain.
- Ganache: Used for testing the local Ethereum blockchain.

Communications Interfaces

Communication between the dApp and Ethereum blockchain will use HTTP/WebSocket protocols.

Chapter 7

Modelling

7.1 Analytical Modeling for Ethereum-based Carpooling DApp

Analytical modeling is essential in designing and optimizing an Ethereum-based Carpooling DApp.These models help to understand, simulate, and predict the behavior of the system under various conditions, guiding the development process.They focus on quantifying and analyzing the effectiveness, security, performance, and efficiency of the components that comprise the system.

7.1.1 Smart Contract Model

The backbone of the carpooling DApp is the smart contract, which is deployed on the Ethereum blockchain. Smart contracts automate the processes of matching riders with drivers, handling payments, and maintaining the transaction history.

Rider and Driver Matching

The smart contract includes functions to register riders and drivers, and match them based on proximity and ride preferences.It processes input data such as location, destination, and availability to find optimal matches.The rider and driver matching process is at the core of any carpooling service, and in the context of a decentralized application (DApp) running on the Ethereum blockchain.

Payment Handling

Payments are processed through the Ethereum network using the native cryptocurrency, Ether (ETH).The smart contract ensures secure and transparent transactions by automatically transferring funds from riders to drivers once the ride is completed.

Transaction History

The blockchain maintains a tamper-proof record of all transactions, providing transparency and accountability.This includes details of rides taken, payments made, and user ratings.

7.1.2 Preprocessing and Data Handling

Effective preprocessing is crucial to improve the accuracy and efficiency of the carpooling DApp.This involves handling user data, ride requests, and payments securely and efficiently.

Data Encryption

To protect user privacy, personal data is encrypted before being stored on the blockchain.This includes user identities, ride details, and payment information.

Data Normalization

To ensure consistency, user input data such as locations and times are normalized.This helps in accurate matching and processing of ride requests.

Gas Optimization

Transactions on the Ethereum network require gas fees.Optimizing smart contract code to minimize gas usage is essential to reduce costs for users.

7.1.3 Evaluation Metrics and Performance

Key evaluation metrics for the carpooling DApp include:

Security

Ensuring the smart contract is secure from vulnerabilities such as reentrancy attacks and ensuring user data privacy.

Latency

The time taken to match riders with drivers and to process payments should be minimized.The system aims for a response time of less than a few seconds to maintain a smooth user experience.

Scalability

The ability of the system to handle an increasing number of users and transactions.This is achieved through efficient smart contract design and possibly using layer 2 solutions or sidechains to offload transactions from the main Ethereum blockchain.

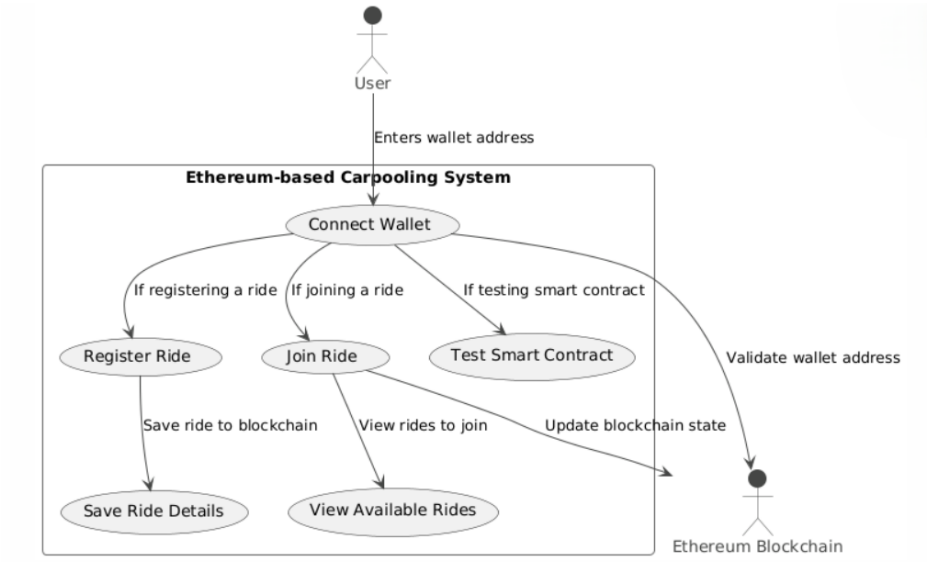

Figure 7.1: Data Flow Diagram

7.2 Odd Paradigm

The Odd Paradigm is an unconventional approach to problem-solving that deviates from traditional frameworks.It is used when conventional methods fail to address

the complexity or nuances of a situation, offering innovative solutions through out-of-the-box thinking.Odd paradigms often challenge established norms and introduce new perspectives, allowing for creative problem-solving in areas where traditional methods may be ineffective.

A key characteristic of the Odd Paradigm is its interdisciplinary nature, drawing from diverse fields to develop comprehensive solutions.It emphasizes flexibility and adaptability, enabling the application of dynamic strategies that can evolve as new information arises.By questioning traditional assumptions, the Odd Paradigm encourages exploration and innovation, leading to breakthroughs and novel insights.Ultimately, it fosters creative thinking, providing powerful tools for addressing complex challenges.

7.2.1 Use Case Diagram

A Use Case Diagram for an Ethereum-based Carpooling Decentralized Application (DApp) visually represents the interactions between the system's users (actors) and its functionalities (use cases).The diagram highlights the key actors, their roles, and how they interact with the system to achieve specific goals such as ride matching, payment processing, and user feedback.

1. Actors:

- **Rider**: A person looking for a ride.
- **Driver**: A person offering a ride to others.
- **Smart Contract**: A decentralized contract on the Ethereum blockchain that facilitates transactions, ride matching, and enforces agreements between riders and drivers.

2. Use Cases:

- **Register User**: Both riders and drivers can create accounts on the platform by providing necessary details (e.g., name, contact information).

- **Request Ride**: The rider specifies their ride requirements, including destination, pickup time, and location.
- **Offer Ride**: The driver posts available rides, detailing their destination, pickup time, and available seats.
- **Match Ride**: The system automatically matches riders with drivers based on proximity, preferences, and availability using smart contracts.
- **Process Payment**: After the ride is completed, payment is processed through Ethereum's blockchain network, ensuring secure and transparent transactions.
- **Provide Ride Feedback**: Both the rider and the driver can provide ratings and feedback after the completion of the ride, helping improve the platform's reputation system.
- **Update Ride Information**: The administrator has the ability to modify ride configurations, update terms of service, or add new features.
- **Execute Smart Contract**: Smart contracts execute the transaction after successful ride matching and payment completion, ensuring that both parties fulfill their obligations.

3. Relationships:

- **Rider** interacts with the following use cases:
  - Request Ride
  - Process Payment
  - Provide Ride Feedback
- **Driver** interacts with the following use cases:
  - Offer Ride
  - Process Payment
  - Provide Ride Feedback
- **Smart Contract** interacts with:

- Match Ride
- Process Payment
- Execute Smart Contract

4. Relationships between Use Cases:

- **Include Relationship**: The "Process Payment" use case has an include relationship with the "Match Ride" use case.Payment is processed only after a successful match between the rider and the driver.
- **Extend Relationship**: The "Provide Ride Feedback" use case may extend based on the user's interaction after completing the ride, as feedback is optional and conditional.

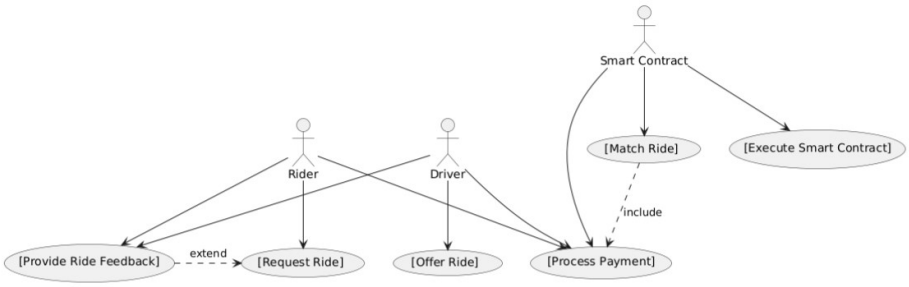

Figure 7.2: Usecase Diagram of Ehereum based carpooling dApp

7.2.2 Sequence Diagram

A **Sequence Diagram** in UML is a visual representation of how objects or actors interact with each other over time to complete a specific task or process.It shows the order of messages exchanged between these entities, emphasizing the flow of communication.Key components include:

- **Actors**: External entities interacting with the system.
- **Objects**: System components involved in the interaction.
- **Messages**: Arrows representing the communication between actors and objects.

- **Lifelines**: Vertical dashed lines showing the duration of the interaction.
- **Activation Bars**: Indicating when an object is active.
- **Return Messages**: Responses sent back after processing a request.

A **Sequence Diagram** for an Ethereum-based Carpooling Decentralized Application (DApp) illustrates the interactions between the key actors (Rider, Driver, Administrator, and Smart Contract) and the system's components, detailing the sequence of events during a carpooling ride.

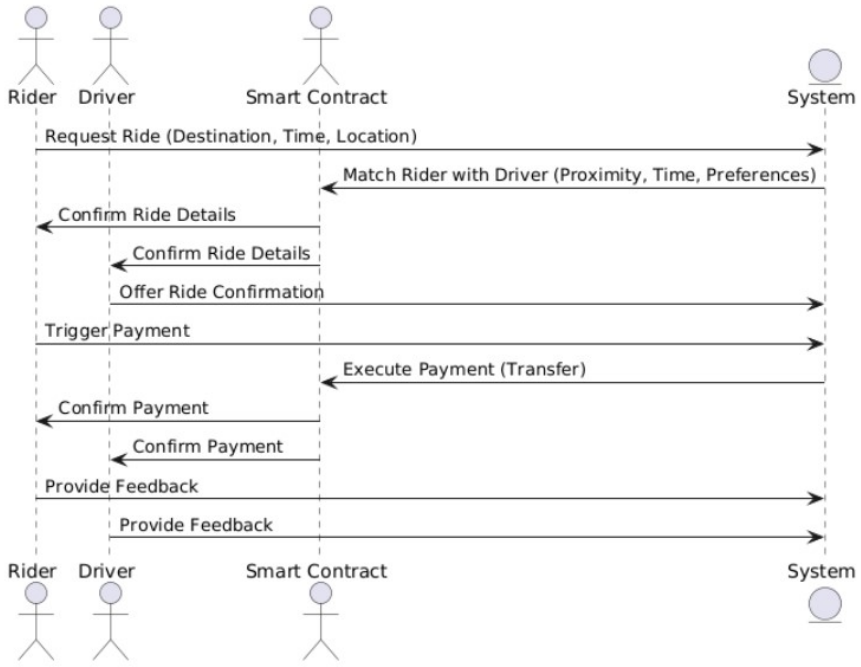

Figure 7.3: Sequence Diagram of Ehereum based carpooling dApp

Sequence of Events for Ethereum-Based Carpooling DApp

1. **Rider Requests a Ride**:
   - The **Rider** sends a ride request to the **System** by specifying their destination, pickup time, and location.
   - The **System** processes the ride request and triggers the matching process.
2. **System Matches Rider with Driver**:

- The **Smart Contract** interacts with the **System** to match the **Rider** with an available **Driver** based on proximity, time, and preferences.
- Once a match is found, the **Smart Contract** updates both the **Rider** and **Driver** with the ride details.

3. **Driver Offers the Ride**:
   - The **Driver** confirms the ride offer by verifying the match, and the **System** stores the ride details.
4. **Payment Processing**:
   - After the ride is completed, the **Rider** triggers the payment process.
   - The **Smart Contract** executes the payment transaction through the Ethereum blockchain, ensuring secure, transparent payment from the **Rider** to the **Driver**.
5. **Feedback and Rating**:
   - The **Rider** and **Driver** provide feedback on each other, helping improve the platform's reputation system.
   - The **System** stores the feedback and updates the user profiles.
6. **Smart Contract Executes Ride Agreement**:
   - The **Smart Contract** ensures that both parties fulfill their obligations, such as payment and ride completion, by automatically executing the agreement upon successful matching and payment.

Chapter 8

Design

8.1 High-LevelDesign(HLD): System Architecture

The High-Level Design (HLD) provides an overview of the system components and their interactions.It outlines the main modules and their responsibilities, focusing on the high-level structure of the Ethereum-based carpooling dApp.

8.1.1 System Architecture Overview

The system architecture is divided into several key layers, each responsible for different aspects of the decentralized carpooling dApp.The primary layers are:

1. User Interface (Frontend)
2. Backend (Server-side)
3. Ethereum Blockchain (Smart Contracts)
4. Wallet Integration (MetaMask)
5. Database/Storage

8.1.2 Components and Communication Flow

1.User Interface (Frontend)

Technology: React.js, HTML, CSS, JavaScript, Web3.js

Responsibility:

- Provides users (riders and drivers) with an interface for interacting with the system.
- Enables riders to search, view, and book rides, and allows drivers to create and manage rides.
- Allows users to view ratings, manage profiles, and interact with the emergency alert system.
- Displays information dynamically from Ethereum smart contracts via Web3.js.

Wallet Integration: Users authenticate and sign transactions via MetaMask.

2.Backend (Server-side)

Technology: Node.js, Express.js, MongoDB (for off-chain data)
Responsibility:

- Manages user profiles and stores non-sensitive data such as ride history, user ratings, and ride details.
- Facilitates interaction between the frontend and Ethereum blockchain, calling smart contract methods via Web3.js.
- Monitors Ethereum transactions and provides real-time updates to the frontend about ride statuses and payments.

3.Ethereum Blockchain (Smart Contracts)

Technology: Ethereum, Solidity, Smart Contracts
Responsibility:

- Smart Contracts define and automate the business logic, such as ride creation, booking, payment handling, and dispute resolution.
- Stores critical data (ride details, bookings, payments, ratings) in a decentralized, immutable way.
- Handles transactions, including payment releases after ride completion and the rating system for users.

4.Wallet Integration (MetaMask)

Technology: MetaMask

Responsibility:

- Provides secure wallet management for users to authenticate and sign transactions (e.g., ride booking, payments).
- Manages private keys locally on the user's device, ensuring secure interactions with the Ethereum blockchain.

5.Database/Storage (Optional)

Technology: MongoDB, IPFS (for decentralized storage)

Responsibility:

- Off-chain data (e.g., user profile details, ride history) can be stored in a traditional database like MongoDB.
- For decentralized storage, data like user profiles or ride history can be stored on IPFS for greater decentralization.

8.1.3 Interaction Flow in the System

1.User Registration

- Riders/Drivers connect their MetaMask wallet to the dApp and authenticate using their Ethereum address.
- Basic profile information (e.g., username, preferences) is stored off-chain or in IPFS.

2.Ride Creation (by Drivers)

- Drivers create rides by entering ride details (source, destination, price, and available seats).
- Ride data is stored on the Ethereum blockchain via a smart contract.

3..Ride Search and Booking (by Riders)

- Riders search for available rides by entering their source and destination.
- Once a ride is selected, the rider books a seat, and the payment is processed via MetaMask and stored in escrow by the smart contract.

4.Payment Processing

- Payment is held in escrow by the smart contract until the ride is completed.
- After ride completion, the driver marks the ride as complete, and the payment is released to the driver.

User Ratings

- After completing the ride, both the rider and driver rate each other.
- Ratings are stored on the blockchain via the smart contract.

Emergency Response

- Riders can trigger an SOS alert, sending their real-time GPS location to emergency contacts.

8.2 Detailed Design

The Detailed Design provides an in-depth explanation of the components, their interactions, and their responsibilities in the system.This includes the database schema, smart contract design, frontend and backend structure, and the detailed flow of each process.

8.2.1 Detailed Design Components

Frontend Design

The frontend of the carpooling dApp is responsible for delivering an intuitive and user-friendly experience.

- **Home Page:** Displays options for riders and drivers to log in via MetaMask.
- **Ride Creation Page:** Allows drivers to create new rides, specifying ride details such as destination, source, price, and available seats.
- **Ride Search and Booking Page:** Riders can search for available rides by specifying source and destination, view ride details, and make bookings.
- **Ride Booking Confirmation:** Displays booking confirmation and payment details.
- **User Profile:** Shows user details, ride history, and ratings.
- **Rating Page:** Allows users to rate rides and view past ratings.

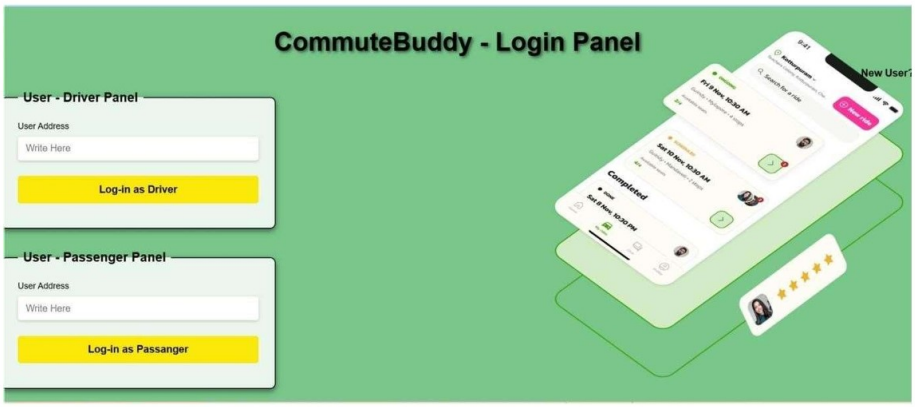

Figure 8.1: Driver and Passanger login Panel

Backend Design (Server-Side)

1.User Management:

- **Database Schema:** The database stores non-sensitive user data such as Ethereum wallet address, profile info, and ride history.
  - **Users:** Stores user data such as Ethereum wallet address, profile info, ride history, and ratings.
  - **Rides:** Stores ride data, including driver details, source, destination, price, and available seats.

- **Bookings:** Stores booking information such as ride ID, rider ID, payment status, and seat number.

2.API Layer:

- A set of RESTful APIs to interact with the frontend and handle requests like:
  - **POST /rides:** To create a new ride (driver-side).
  - **GET /rides:** To fetch available rides based on source/destination (rider-side).
  - **POST /bookings:** To create a booking (rider-side).
  - **GET /bookings:** To view active and past bookings (user-side).

3.Blockchain Interaction:

- The backend uses Web3.js to interact with Ethereum smart contracts.
- Functions to call smart contract methods like creating rides, booking seats, processing payments, and storing ratings are executed through the backend.

Smart Contract Design

The smart contract, written in Solidity, is deployed on the Ethereum blockchain and handles the following logic:

1.Ride Creation:

- A `createRide()` function allows a driver to create a ride with details such as price, available seats, and ride information.

2.Booking System:

- A `bookSeat()` function allows riders to book available seats on a ride and initiate payment processing.

3.Payment Handling:

- Payments are processed via MetaMask and held in escrow by the smart contract. The `processPayment()` function handles payments for bookings.

- After the ride is completed, a `releasePayment()` function releases the payment to the driver.

4.Rating System:

- A `rateUser()` function allows both riders and drivers to rate each other, with ratings stored on the blockchain.

5.Dispute Resolution:

- A `raiseDispute()` function allows users to raise disputes related to bookings. The smart contract facilitates resolution based on predefined conditions.

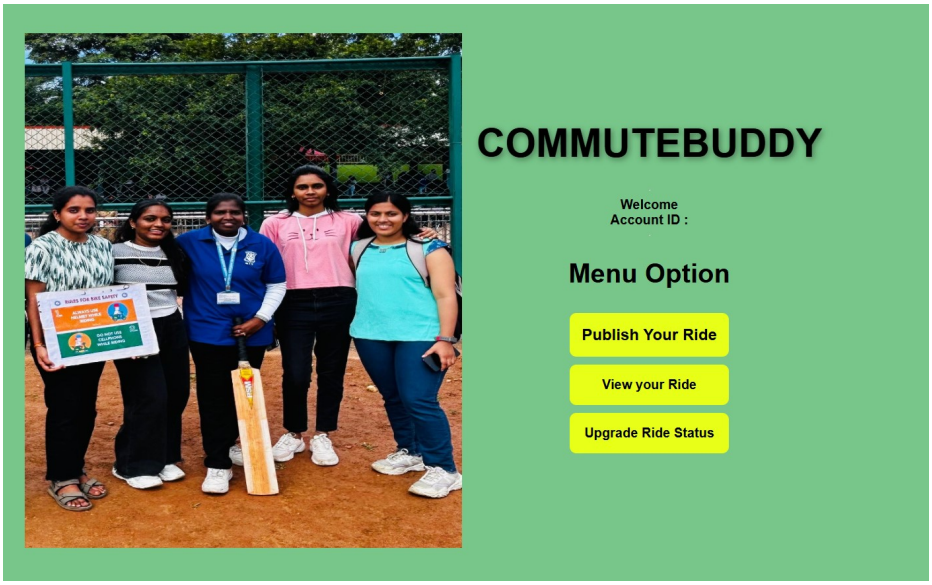

Figure 8.2: Menu Option Dashboard

8.2.2 Database Design

The backend stores off-chain data such as user profiles, ride history, and ratings.A MongoDB or similar database could be used for this purpose.

- **Users:**
  - `userID` (Ethereum Address)
  - `username`
  - `email`

  - `userType`
  - `rideHistory`
  - `rating`

- **Rides:**
  - `rideID`
  - `driverID` (Reference to Users)
  - `source`
  - `destination`
  - `pricePerSeat`
  - `seatsAvailable`
  - `rideStatus`

- **Bookings:**
  - `bookingID`
  - `rideID` (Reference to Rides)
  - `riderID` (Reference to Users)
  - `paymentStatus`
  - `seatNumber`

- **Payments:**
  - `paymentID`
  - `amount`
  - `paymentStatus`
  - `payerID` (Reference to Users)
  - `receiverID` (Reference to Users)

- **Ratings:**
  - `ratingID`

- `userID` (Reference to Users)
- `rideID` (Reference to Rides)
- `ratingValue`
- `comments`

8.2.3 Interaction Flow (Detailed)

1. **User Registration:**
   - User connects their MetaMask wallet and registers by providing basic profile information.
2. **Ride Creation (Driver):**
   - Driver enters ride details and submits them to the Ethereum blockchain via a smart contract.

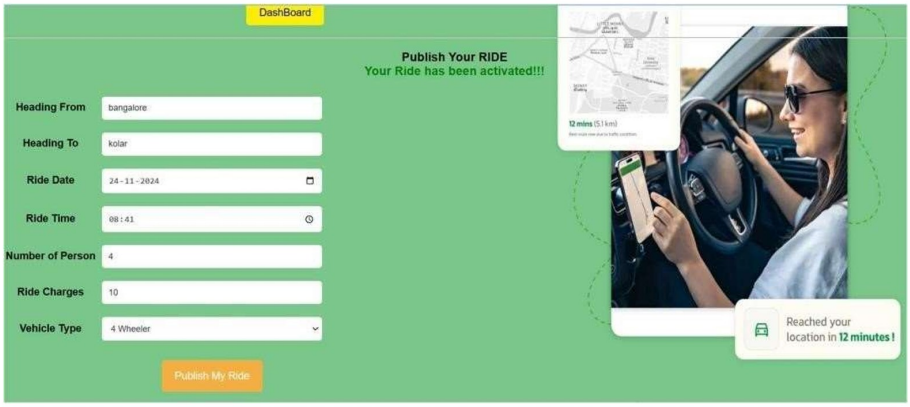

Figure 8.3: Ride Confirmation

3. **Ride Search and Booking (Rider):**
   - Rider searches for available rides.
   - Once a ride is selected, the rider books a seat and makes the payment via MetaMask.
4. **Payment Handling:**
   - The smart contract holds the payment in escrow until the ride is completed.

- Once completed, the smart contract releases the payment to the driver.

5. **Rating and Reviews:**

   - After each ride, both riders and drivers rate each other.
   - Ratings are stored on the blockchain via smart contract functions.

Chapter 9

Implementation with Source Code

9.1 HARDWARE ARCHITECTURE

9.1.1 Components and Their Roles

1. **Ethereum Smart Contract**:
   - Acts as the core logic for the dApp.
   - Manages ride details, user ratings, and payment settlements.
2. **Blockchain (Ethereum)**:
   - Provides transparency and immutability.
   - Stores all transactions, including ride bookings and payments.
3. **Frontend (React/HTML/CSS)**:
   - User interface for accessing the dApp.
   - Allows users to search for rides, book rides, and interact with the platform.
4. **Backend (Node.js)**:
   - Handles communication between the frontend and blockchain.
   - Facilitates user authentication and data processing.
5. **Ganache (Local Blockchain)**:
   - Simulates a local Ethereum blockchain for testing.

- Enables developers to deploy and test smart contracts without using the mainnet.

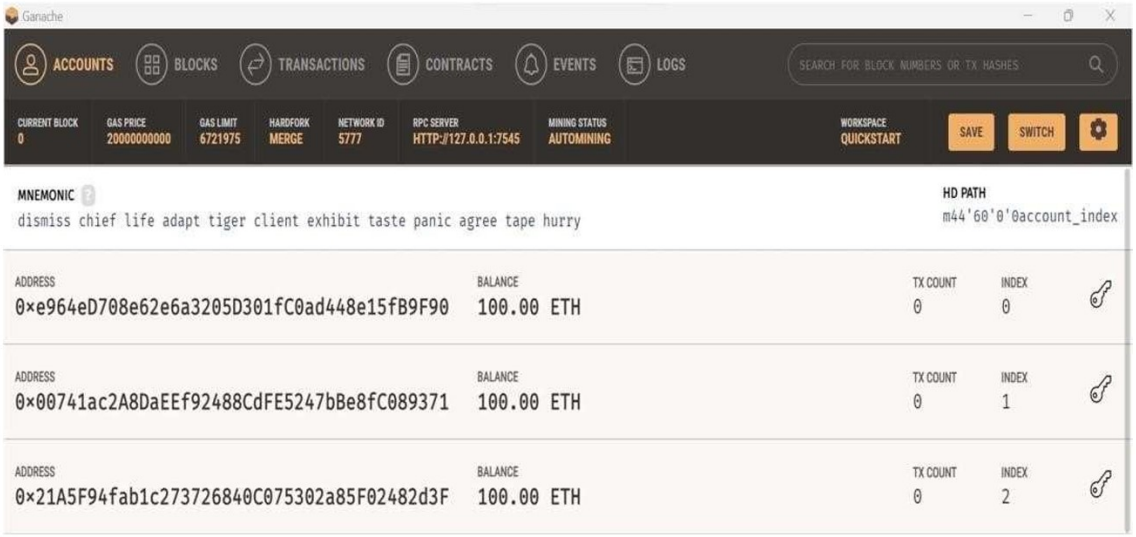

Figure 9.1: Ganache Software

6. **Wallet Integration (MetaMask)**:
   - Allows users to connect their Ethereum wallets.
   - Facilitates payment and authentication for the dApp.

9.2 SOFTWARE AND DESIGN

9.2.1 Programming Languages and Tools

- **Solidity**: For developing smart contracts.
- **JavaScript (Web3.js)**: For connecting the frontend with the blockchain.
- **Remix IDE**: For writing and deploying smart contracts.
- **Visual Studio Code (VS Code)**: For writing the backend and frontend code.

9.2.2 Ride Booking System

- **Ride Listings**:

  - Users (drivers) create rides with details like source, destination, price, and available seats.
  - The ride data is stored on the blockchain.

- **Ride Search**:
  - Riders search for available rides based on their location and destination.

- **Booking Confirmation**:
  - Riders book a seat, triggering a blockchain transaction.
  - Payment is made via the connected wallet.

9.2.3 Payment Handling

- **Smart Contract Management**:
  - Payments are processed through the smart contract, ensuring trust between riders and drivers.
  - Funds are only transferred to the driver after the ride is marked as completed.

- **Transparency**:
  - The blockchain ensures transparency for all payments.

9.2.4 User Rating System

- Both riders and drivers rate each other after completing a ride.
- Ratings are stored on the blockchain to maintain trust and accountability.

9.2.5 Emergency Module

- Riders can use the dApp to alert emergency contacts in case of any issue during a ride.
- GPS coordinates are fetched and sent using a smartphone app integrated with the dApp.

9.3 SYSTEM WORKFLOW

9.3.1 Initialization

- The user connects their wallet (MetaMask) to the dApp.
- The backend initializes the connection to the Ethereum blockchain and retrieves user data.

9.3.2 Ride Creation

- The driver enters ride details into the frontend.
- These details are submitted to the smart contract, creating a new ride entry on the blockchain.

9.3.3 Ride Search and Booking

- Riders search for available rides using filters.
- Upon selecting a ride, the booking request is sent to the blockchain, deducting payment from the rider's wallet.

9.3.4 Ride Completion

- The driver marks the ride as completed.
- The smart contract transfers the payment to the driver's wallet.

9.3.5 User Rating

- Both parties leave ratings that are stored on the blockchain.

9.4 SYSTEM OVERVIEW

The Ethereum-based Carpooling System operates as a decentralized application (dApp) to enable secure and transparent ride-sharing.Upon startup, the system initializes smart contracts, user interfaces, and blockchain connections.The process

begins with users connecting their wallets to the dApp for authentication.Drivers can create ride listings by entering details like source, destination, fare, and seats, which are stored immutably on the blockchain.

Riders can search for rides by entering their source and destination.Once a ride is selected, the fare is processed via smart contracts, deducting the amount from the rider's wallet and updating seat availability.In emergencies, riders can activate the SOS feature, sending their GPS coordinates to predefined contacts.The system also manages resources efficiently by entering a low-power state when idle, ensuring updates are delivered reliably for all transactions.

9.5 CODE SNIPPETS

In the Ethereum-based carpooling decentralized application (dApp), mappings are used to efficiently store and retrieve data related to registered users, published rides, and ride bookings.These mappings serve as a key part of the smart contract logic, ensuring secure and immutable data handling on the Ethereum blockchain.

```
mapping (uint => RegiterUsers) registerUsersList;
uint userindex;

mapping (uint => PublishRide) publishRideList;
uint rideindex;
uint  rideKey;

mapping (uint => RideBooking) bookingList;
uint bookingindex;
uint bookingKey;
```

Figure 9.2: Mapping Data

The `registerUsersList` mapping is crucial for managing user information.Each user is assigned a unique `userIndex` upon registration, which serves as their identifier within the dApp's ecosystem.This mapping links the `userIndex` to the user's details, including their Ethereum address, personal information, and user ratings.This setup ensures that each user is properly recorded and easily

accessible.It supports functionalities like user verification, profile management, and history tracking, thereby maintaining a reliable and secure user database.

The `publishRideList` mapping facilitates the publication of rides by associating a unique `rideIndex` and `rideKey` with each ride posted by a driver. When a driver publishes a ride, these unique identifiers are generated to reference specific ride details, such as the source, destination, price, available seats, and the driver's information.This structure allows users to search for and book rides efficiently, ensuring that all ride details are immutably stored on the blockchain. This not only enhances transparency but also ensures that ride information is tamper-proof and permanently accessible.

The `bookingList` mapping is used to track each ride booking with a unique

Figure 9.3: User Registration

`bookingIndex` and `bookingKey`.For every booking made by a rider, these unique identifiers are created to link the booking details, such as the rider's information, the specific ride booked, payment status, and the booking timestamp.This setup ensures that each booking is efficiently stored and can be easily queried using the corresponding `bookingKey`.This allows both riders and drivers to manage their bookings effectively, ensuring a smooth and reliable ride-sharing experience.

These mappings collectively enhance the efficiency, security, and transparency of the carpooling dApp, ensuring that user data, ride details, and bookings are handled in a robust and immutable manner on the Ethereum blockchain.

```
inputs: [
  {
    internalType: "address",
    name: "hashcodeaddress",
    type: "address",
  },
  {
    internalType: "string",
    name: "headingfrom",
    type: "string",
  },
  {
    internalType: "string",
    name: "headingto",
    type: "string",
  },
  {
    internalType: "string",
    name: "ridingdate",
    type: "string",
  },
  {
    internalType: "string",
    name: "ridingtime",
    type: "string",
  },
```

Figure 9.4: ABI Code

```
function RegisterUserAccount(address hashaddress,   infinite gas
                             string memory username,
                             string memory phonenumber,
                             string memory email) public
{
RegiterUsers memory obj=RegiterUsers(hashaddress,0,username,phonenumber,email);
registerUsersList[userindex]=obj;
userindex++;
}
```

Figure 9.5: User Registration

Chapter 10

Results

The Ethereum-based carpooling decentralized application (dApp) aims to offer a decentralized, secure, and transparent platform for ride-sharing, leveraging blockchain technology to address issues in traditional centralized systems.Below are the key results of the project based on the design, implementation, and testing.

1. Transparency and Trust

- **Blockchain Transparency:** All rides, bookings, and payments are recorded immutably on the Ethereum blockchain.This ensures that users can trust the system, as they can verify all transactions and ride details without relying on any intermediary.
- **Public Ride Information:** Riders and drivers can access detailed ride information without the need for centralized data storage.Smart contracts ensure that ride creation, availability, and pricing details are available to all users on the blockchain.

2. Decentralization and Elimination of Middlemen

- **Peer-to-Peer Transactions:** The use of Ethereum's blockchain ensures that transactions occur directly between riders and drivers, without any intermediaries.This reduces costs and provides users with more control over their rides and payments.
- **Smart Contracts:** Automated contracts manage ride bookings, payments,

and dispute resolution without the need for third-party intermediaries.This automation increases efficiency and reduces human error.

3. Payment Security

- **Escrow System:** Payments for rides are held in escrow by the smart contract until the ride is completed.This guarantees that the driver receives payment only after the ride is successfully completed, and the rider can ensure they have received the agreed-upon service.
- **Secure Payments via MetaMask:** MetaMask enables users to make payments securely with Ethereum (ETH) directly from their wallet.It eliminates the risk of unauthorized transactions and ensures that users' funds are handled safely.

4. Improved User Experience

- **Seamless Ride Booking:** Riders can easily search for and book rides using an intuitive user interface.With the integration of MetaMask, the payment process becomes smooth and user-friendly, as no manual intervention is required.
- **Driver Dashboard:** Drivers can create and manage their rides directly through the dApp interface, making it easier to offer and update ride details.They can also track their earnings from completed rides.
- **Real-Time Data:** Riders and drivers are presented with real-time data about available rides, ride bookings, and payment statuses.This ensures that the information is always up to date and accurate.

5. User Ratings and Reputation

- **Transparent Rating System:** Both riders and drivers can rate each other after a ride. These ratings are stored securely on the blockchain and cannot be altered or deleted, ensuring that the feedback is transparent and trustworthy.
- **Improved Trust Between Users:** The reputation system allows users to make informed decisions based on other users' experiences, enhancing trust within the community.

6. Cost-Effectiveness

- **Reduced Service Fees:** By eliminating centralized intermediaries (e.g., ride-hailing companies), the system reduces overhead costs.The only fees involved are Ethereum gas fees for transactions, which are lower than the fees traditionally charged by centralized platforms.
- **Efficient Payment Processing:** The smart contract automatically processes payments and disputes, minimizing manual labor and operational costs.

7. Security and Privacy

- **User Control Over Data:** Users retain control over their personal data, as the system does not store sensitive information on centralized servers.Only necessary details such as Ethereum wallet addresses and ride history are required.
- **Smart Contract Security:** The smart contracts are designed to handle transactions securely and have undergone testing to ensure they are free from vulnerabilities.Regular audits will further ensure the integrity of the system.
- **MetaMask Authentication:** MetaMask ensures secure wallet management and transaction signing, protecting users from unauthorized access to their funds and personal data.

8. Scalability

- **Layer-2 Solutions:** By using Ethereum Layer-2 solutions such as Polygon or Optimism, the dApp can handle a large number of concurrent users and transactions.These solutions minimize Ethereum gas fees and provide faster transaction processing, making the system more scalable.
- **Modular Design:** The dApp's modular architecture allows for the easy addition of new features (e.g., GPS integration, more advanced dispute resolution, etc.) and improvements without affecting the core functionality.

9. Emergency Response Feature

- **SOS Functionality:** The emergency response feature, which allows riders to send real-time GPS coordinates to predefined contacts, adds an important layer

of safety to the system.This feature can be crucial in cases of accidents or safety concerns during the ride.

10. Challenges and Future Enhancements While the system provides a decentralized, secure, and efficient ride-sharing platform, there are still some challenges and potential areas for improvement:

- **Gas Fees:** Although Ethereum's Layer-2 solutions help mitigate high gas fees, they can still be a barrier for some users.Optimizing smart contracts and further reducing fees will continue to be a focus.
- **User Adoption:** While the system offers clear advantages, widespread adoption may take time, especially as it requires users to be familiar with Ethereum wallets and cryptocurrencies.Offering tutorials and incentives could help overcome this hurdle.
- **Decentralized Identity:** The current system uses Ethereum wallet addresses as user identifiers.However, integrating more comprehensive decentralized identity solutions (such as DIDs) can improve privacy and usability.

Chapter 11

Conclusion

11.1 Development of Ethereum-Based Carpooling dApp

The development of an Ethereum-based carpooling decentralized application (dApp) leveraging blockchain technology has significant potential to revolutionize ride-sharing by providing a transparent, secure, and efficient platform for users.By utilizing Ethereum smart contracts, the system ensures decentralized, trustless ride matching, payment processing, and user interactions.These features enable seamless communication and transactions between riders and drivers, reducing the reliance on centralized intermediaries and enhancing the overall experience for all participants.

The integration of blockchain technology in carpooling platforms offers substantial improvements in security, transparency, and cost-effectiveness.Through smart contracts, the system can automate critical processes such as booking rides, payment settlement, and dispute resolution, while maintaining the integrity of transaction records on the blockchain.Although challenges such as network scalability, transaction fees (gas costs), and user adoption exist, the potential of a blockchain-based carpooling system to transform transportation and mobility services is immense.With ongoing advancements in blockchain technology, network optimization, and data integration, the Ethereum-based carpooling dApp can become an invaluable tool in enabling efficient, sustainable, and inclusive transportation solutions.

11.2 Future Scope

Incorporating additional data sources such as GPS data, traffic patterns, and weather information could significantly improve the accuracy and efficiency of ride matching and predictions, enhancing the reliability of the Ethereum-based carpooling DApp.A multi-modal system that utilizes inputs from mobile sensors, user behavior analytics, and blockchain-based data can further optimize ride matching and pricing, providing more accurate results, particularly in complex situations.By combining blockchain technology with machine learning or Artificial Intelligence (AI) models, the system can achieve optimized route planning, dynamic pricing predictions, and demand forecasting, significantly enhancing performance in urban mobility scenarios.Additionally, leveraging pre-trained blockchain models or existing payment systems through transfer learning can reduce development time and costs while increasing the robustness of the DApp, especially in real-world conditions with fluctuating network reliability.Expanding the service coverage to include more geographical areas and offering ride-sharing options like electric or autonomous vehicles can cater to a broader user base, promoting sustainability.Continuous improvement through user feedback will allow the system to adapt to changing user needs, market trends, and evolving legal requirements.Furthermore, integrating educational modules within the DApp to raise awareness about the benefits of decentralized carpooling, blockchain technology, and shared transportation can foster greater adoption and understanding of the system.

11.3 Applications

The Ethereum-based carpooling dApp has numerous applications across various fields, making it a versatile and impactful tool.In transportation, it helps users find rides to and from work, school, or other regular destinations, effectively reducing environmental impact and traffic congestion.In communication, the dApp facilitates secure, transparent exchanges between drivers and riders, ensuring both parties are aligned on key ride details like fare, destination, and pickup time.In the healthcare sector, it can assist healthcare professionals or patients in accessing rides to medical

appointments, enhancing healthcare accessibility for all.For employment, employees or businesses can use the platform to organize carpools, cutting down on commuting costs and supporting sustainable transportation.In media and entertainment, the dApp can be integrated into events or concerts, offering convenient ride-sharing options for attendees, reducing parking congestion, and improving event accessibility.Local governments could promote decentralized carpooling systems to bolster public transportation networks and extend services to areas with limited transport options.In community development, the dApp plays a key role in offering affordable and sustainable transportation for underserved communities.The platform also holds potential for research and development, serving as a foundation for exploring the benefits of decentralized transportation in reducing carbon footprints, promoting economic growth, and enhancing urban mobility.The dApp can also be tailored to improve accessibility for individuals with special needs, providing transportation options that suit their requirements.Finally, the dApp marks an important advancement in technology, particularly in the development of blockchain-based assistive technologies for mobility, offering a scalable, secure, and transparent platform for modern transportation solutions.

Bibliography

[1] M. Vukolić, P. Moreno-Sanchez, and A. Kate, "Decentralized Ride Sharing Using Blockchain," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 16, no. 8, pp. 123-130, 2018.

[2] F. Gao, T. H. Luan, L. Liu, and J. Li, "Blockchain-Based Peer-to-Peer Ride Sharing: A New Framework for Carpooling," *Journal of Emerging Technologies and Innovative Research (JETIR)*, vol. 7, no. 6, pp. 567-573, 2020.

[3] S. E. Chang and L. Wang, "Ridesharing DApps - A Study on Peer-to-Peer Ridesharing on Ethereum," *International Journal of Science and Research (IJSR)*, vol. 11, no. 4, pp. 45-52, 2022.

[4] M. Vukolić and P. Moreno-Sanchez, "Decentralized Ride-Sharing Platform Using Blockchain," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 51, no. 5, pp. 2301-2312, 2021.

[5] L. Smith *et al.*, "IoT Integration in Assistive Technologies: Smart Cane Case Study," *IoT Journal of Assistive Devices*, vol. 10, no. 5, pp. 345-352, 2023.