Indian Institute of Technology–Madras (IIT–Madras)

Department of Computer Science and Engineering

M.Tech in Computer Science & Engineering

CS6858: Distributed Trust

AutoTrust - Vehicle Lifecycle Platform

Instructor:
Prof. John Augustine

Teaching Assistant:
Barath Ashok

Students:
Muzaffar Ahmed (CS23M517)
Dharmendra Agrawal (CS23M508)

April 18, 2025

Contents

1 Project Summary

This project implements a decentralized Vehicle Management System on the Hedera Network, leveraging blockchain technology for secure and transparent vehicle registration and management. The system provides a tamper-proof solution for vehicle registration, ownership transfer, and user profile management. By utilizing smart contracts and distributed ledger technology, it ensures data integrity and transparency while eliminating the need for centralized authority.

2 Acknowledgements

We would like to express our gratitude to:

- Prof. John Augustine (CSE Dept., IIT Madras) for his guidance and inspiring work in distributed trust and secure distributed computing
- Barath Ashok (MS Scholar, IIT Madras) for his teaching assistance
- The Hedera development team for their documentation and tools

3 Problem Statement

The vehicle reselling market suffers from several challenges that can be effectively addressed through blockchain technology:

- Centralized authority leading to single points of failure
  - Risk of system downtime
  - Vulnerability to cyber attacks
  - Dependence on single authority
- Lack of transparency in vehicle history
  - Difficulty in verifying past ownership
  - Hidden accident history
- Vulnerability to data tampering
  - Manual record manipulation
  - Unauthorized modifications

  - Lack of audit trail

- Inefficient verification processes
  - Time-consuming manual checks
  - Multiple agency coordination
  - Paper-based documentation

- Limited accessibility and interoperability
  - Geographic restrictions
  - System incompatibility
  - Data silos

4 Motivation & Objectives

4.1 Motivation

The motivation behind this project stems from several key factors:

- Need for a transparent and secure vehicle management system
  - Growing concerns about vehicle-related fraud
  - Increasing demand for transparent transactions
  - Need for reliable vehicle history

- Elimination of fraudulent activities in vehicle registration
  - Prevention of duplicate registrations
  - Reduction in stolen vehicle registrations
  - Elimination of fake documentation

- Decentralization of vehicle ownership records
  - Distributed storage of records
  - Elimination of single point of control
  - Enhanced system resilience

- Enhanced trust in vehicle transactions

  - Verifiable ownership history
  - Transparent transaction records
  - Immutable data storage

4.2 Objectives

The primary objectives of this project are:

- Develop a decentralized vehicle management system
  - Implement blockchain-based architecture
  - Design secure smart contracts
  - Create user-friendly interface
- Implement secure smart contracts for vehicle registration
  - Design efficient data structures
  - Implement access control mechanisms
  - Ensure transaction security
- Create a user-friendly interface for system interaction
  - Design intuitive UI/UX
  - Implement responsive design
  - Ensure cross-platform compatibility
- Ensure data integrity through blockchain technology
  - Implement cryptographic verification
  - Maintain audit trails
  - Ensure data immutability
- Enable seamless vehicle ownership transfers
  - Streamline transfer process
  - Automate verification steps
  - Ensure secure transactions

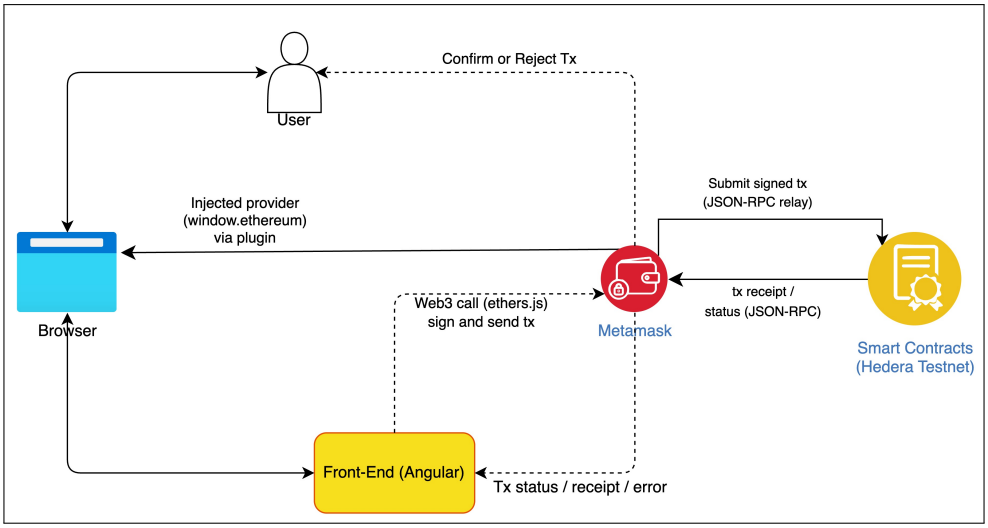


Figure 1: *System Architecture Diagram*

5 System Architecture

The system follows a decentralized architecture with the following components:

5.1 Frontend Layer

- Angular Application
  - User Interface Components
  - Service Layer
  - State Management
- Web3 Integration
  - MetaMask Wallet Connection
  - Transaction Management
  - Event Handling

5.2 Blockchain Layer

- Smart Contracts
  - VehicleLedger Contract
  - UserProfile Contract
  - Access Control

- Hedera Network
  - Consensus Mechanism
  - Transaction Processing
  - State Management

5.3 Storage Layer

- IPFS Storage
  - Document Storage
  - Image Storage
  - Metadata Management
- Local Storage
  - User Preferences
  - Cache Management
  - Session Data

6 Technology Stack

6.1 Frontend

- Angular
  - Component-based Architecture
  - Reactive Forms
  - HTTP Client
  - Router Module
- TypeScript
  - Type Safety
  - Object-Oriented Programming
  - Interface Definitions
- SCSS for styling

  - Modular Styles
  - Responsive Design
  - Theme Support
- ethers for blockchain interaction
  - Contract Interaction
- MetaMask for wallet integration
  - Account Management
  - Transaction Signing

6.2 Backend/Smart Contracts

- Solidity for smart contracts
  - Contract Development
  - Security Best Practices
  - Gas Optimization
- Hedera Network for blockchain
  - Consensus Service
  - Token Service
  - Smart Contract Service
- Node.js for contract deployment
  - Contract compilation and deployment Scripts
- IPFS for distributed storage
  - Document Storage
  - Distributed application deployment

7 Smart Contract Design

The system consists of two main smart contracts. For complete code listings, please refer to Appendix 17.1.

7.1 VehicleLedger Contract

The VehicleLedger contract manages vehicle registration and ownership transfers. Key features include:

- Vehicle registration with owner, registration number and manufacturing year
- Ownership transfer with resale amount tracking
- Maintenance history management
  - Date of maintenance
  - Type of maintenance
  - Service provider details
- Insurance history tracking
  - Insurance reference number
  - Document hash
  - Document link
- Accident history recording
  - Accident date
  - Report document hash
  - Report document link
- Complete vehicle history retrieval
  - Basic details
  - Past owners
  - Maintenance records
  - Insurance records
  - Accident records
  - Resale history

7.2 UserProfile Contract

The UserProfile contract handles user management and authentication. Key features include:

- User profile management
  - Name and phone number updates
  - Vehicle ownership tracking
  - Automatic profile creation
- Vehicle ownership management
  - Add vehicles to user profile
  - Remove vehicles from profile
  - List all owned vehicles
- Profile information retrieval
  - Get user details
  - Get owned vehicles
  - Get wallet address

8 Application Workflow

8.1 User Registration

1. Connect MetaMask wallet
   - Initialize Web3 provider
   - Request account access
   - Verify network connection
2. Create user profile
   - Enter name and phone number
   - Profile automatically created
   - No additional documents required
3. Profile verification
   - Verify wallet address

- Check profile creation
- Confirm successful registration

8.2 Vehicle Registration

1. Submit vehicle details
    - Registration number
    - Year of manufacturing
    - Verify ownership
2. Complete registration
    - Sign transaction
    - Pay registration fee
    - Receive confirmation
3. Add vehicle history
    - Maintenance records
    - Insurance details
    - Accident reports

8.3 Vehicle Transfer

1. Initiate transfer request
    - Enter vehicle registration number
    - Specify new owner address
    - Set resale amount
2. Verify ownership
    - Check current ownership
    - Verify transfer rights

- Validate transaction

3. Complete transfer
    - Sign transfer agreement
    - Process transaction
    - Update ownership records
4. Update profiles
    - Remove from seller's profile
    - Add to buyer's profile
    - Record resale amount

9 UI Components

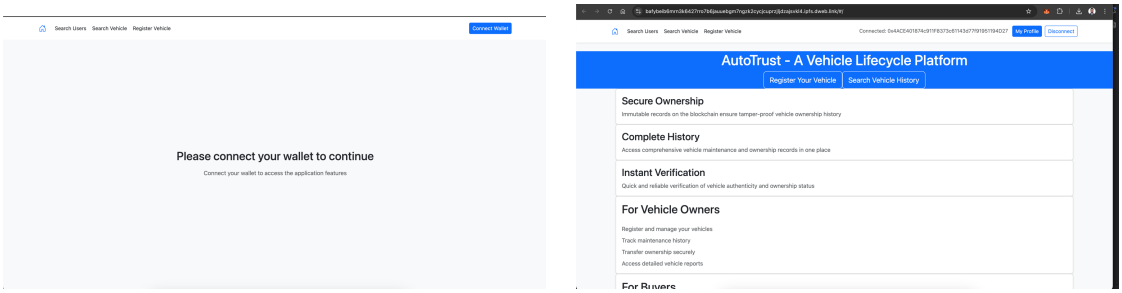
9.1 Core Pages

Page	Key Features
Login & Home Page	Connect wallet; overview; quick actions; navigation tabs
Vehicle Page	View and update maintenance, accident, past owners and insurance details; resale vehicle
Vehicle Registration	New vehicle registration
Search Vehicle	Advanced filters; results display
Search User	Lookup by address; results preview
User Profile	Personal info; document management; activity history

Table 1: *Core UI Pages and Features*

9.2 Login & Home Page

- Ask user to connect MetaMask wallet
- Dashboard overview, quick actions, navigation tabs



(a) *Request to connect Wallet* **(b)** *Home Page*

Figure 2: *Login and Home Page*

9.3 Vehicle Registration

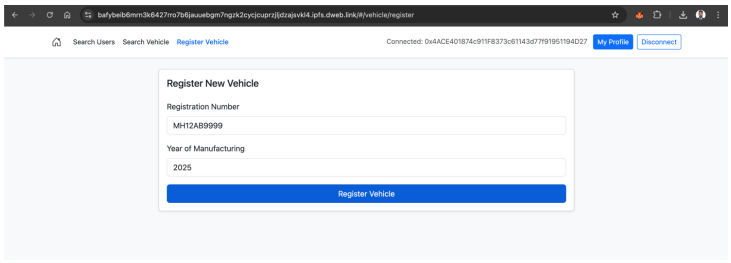
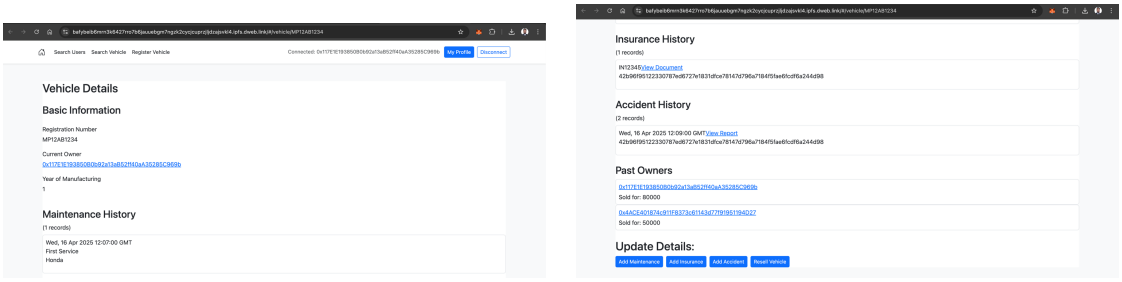


Figure 3: *Vehicle Registration Form*

9.4 Vehicle Page

- View basic details
- View history: maintenance, accident, past owners, insurance
- Buttons to update details and resell vehicle



(a) *Basic Details & Maintenance* **(b)** *Insurance, Accident & Ownership*

Figure 4: *Vehicle Page — Details & History*

9.5 Update Maintenance

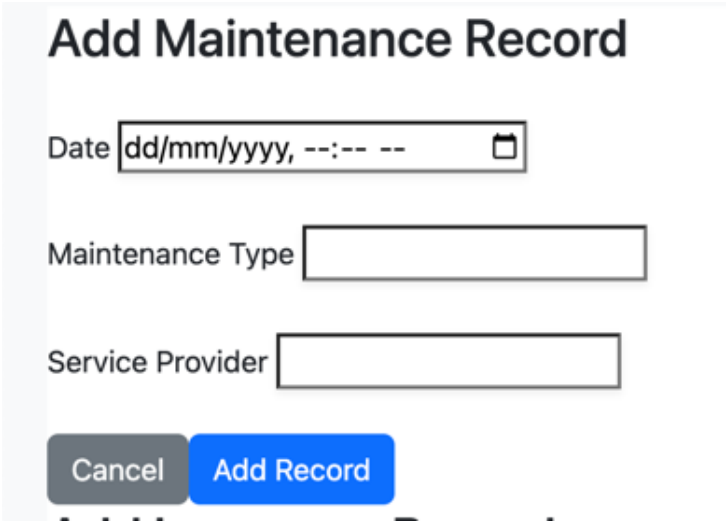


Figure 5: *Add Maintenance Record*

9.6 Update Accident

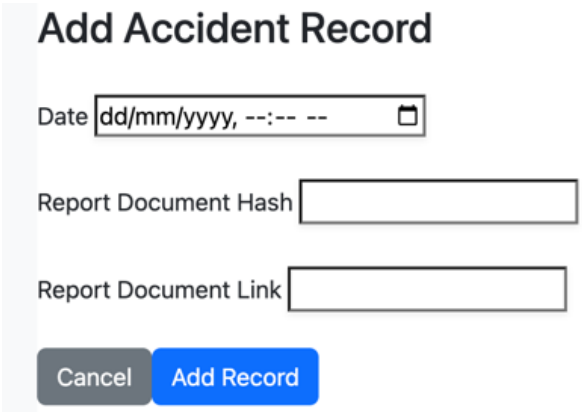


Figure 6: *Add Accident Record*

9.7 Update Insurance

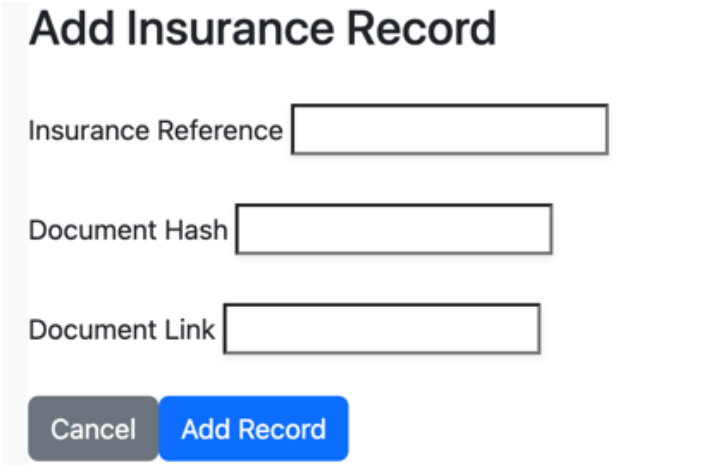


Figure 7: *Add Insurance Record*

9.8 Resell Vehicle

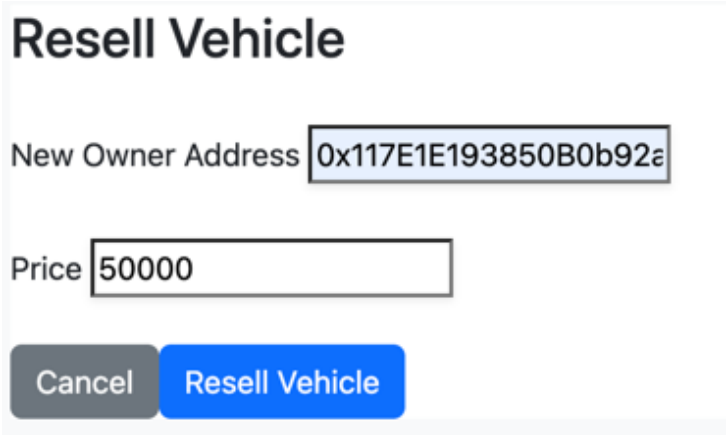


Figure 8: *Vehicle Resale Interface*

9.9 Search Vehicle

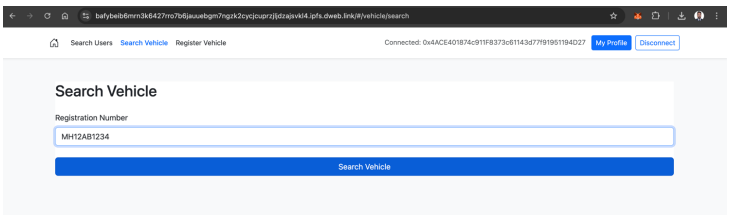


Figure 9: *Vehicle Search Interface*

9.10 Search User

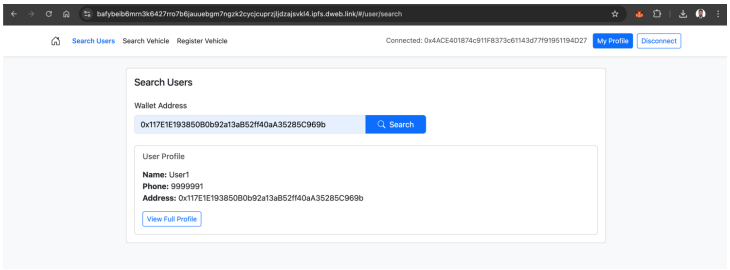


Figure 10: *User Search Interface*

9.11 User Profile

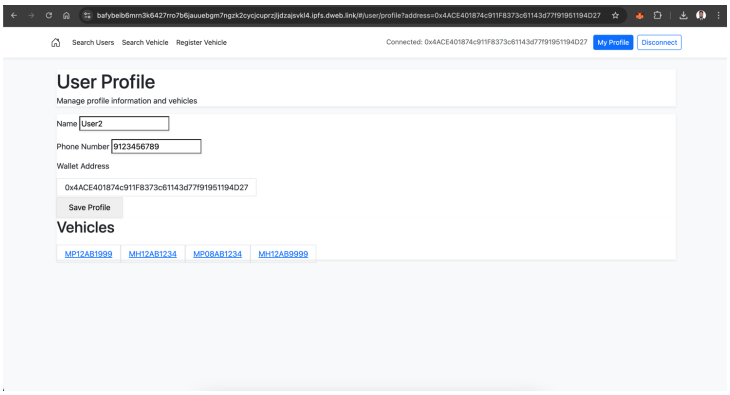


Figure 11: *User Profile Page*

9.12 MetaMask Transaction Request

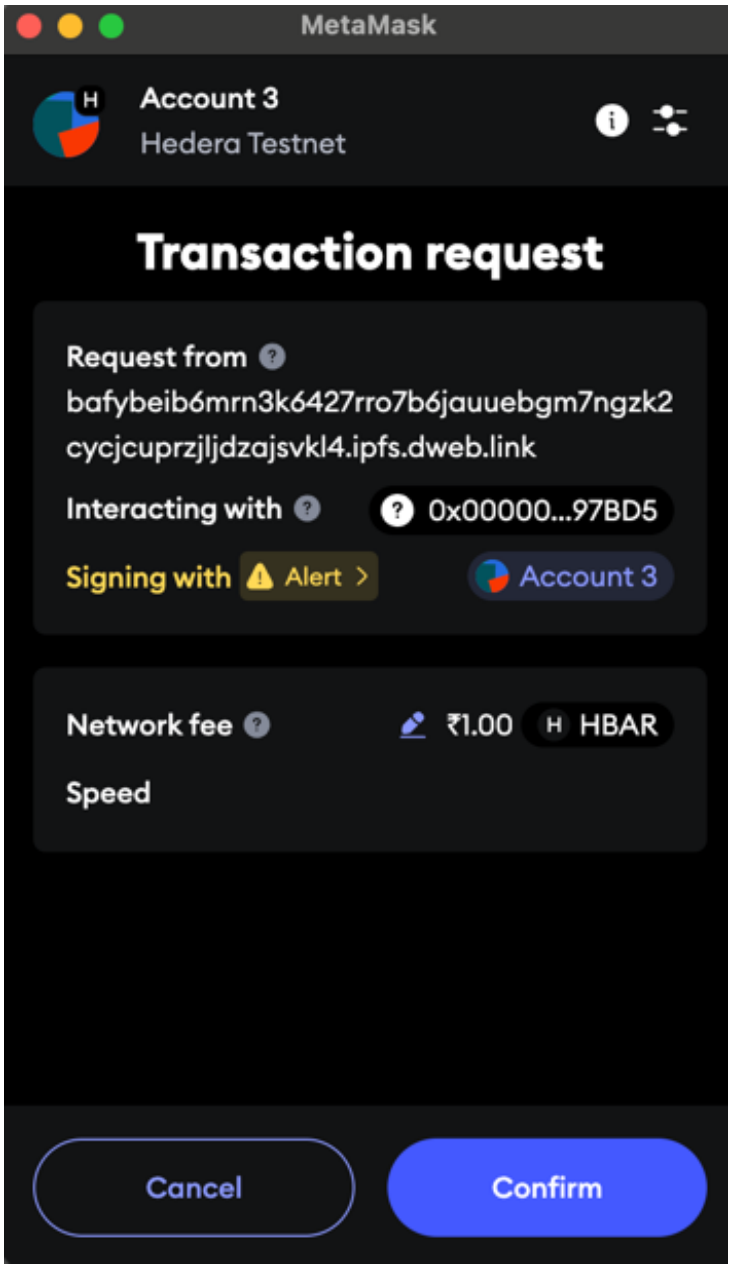


Figure 12: *MetaMask Transaction Request*

10 Sample Interactions

10.1 Vehicle Registration

1. User connects wallet
    - Click ”Connect Wallet”
    - Approve MetaMask popup

    - Verify connection

2. Fills registration form
    - Enter vehicle details
3. Submits transaction
    - Click ”Register”
    - Confirm transaction
    - Wait for confirmation
4. Confirms in MetaMask
    - Review transaction
    - Approve gas fee
    - Sign transaction
5. Receives confirmation
    - View success message
    - Download certificate
    - Check transaction status

10.2 Vehicle Search

1. Enter Registration number
2. Navigate to vehicle page
3. Access detailed information
    - View vehicle details
    - Check ownership history
    - View documents
4. Verify ownership
    - Check current owner

- View transfer history
- Verify authenticity

11 Security & Data Integrity

11.1 Security Measures

- Blockchain-based data storage
  - Distributed ledger
  - Cryptographic hashing
  - Consensus mechanism
- Smart contract access control
  - Vehicle: only current owner can update details or resell a vehicle
  - User: Current user can only update own profile
- Wallet authentication
  - Private key security
  - Transaction signing

11.2 Data Integrity

- Immutable records
  - Blockchain storage
  - Hash verification
  - Timestamp validation
- Transparent transactions
  - Public ledger
  - Transaction history

12 Deployment

12.1 Deployment

The application is deployed on:

- Frontend: IPFS Gateway (please refer to Appendix 18.2)
  - Angular production build
  - Deployment of build files on IPFS using Pinata
- Smart Contracts: Hedera Testnet
  - Contract compilation and deployment
  - Address verification
  - ABI management
- Distributed Storage: IPFS
  - Document storage
  - Distributed Applicatoin deployment

13 Challenges Faced

- Integration with Hedera Network
  - Transaction handling
  - Gas optimization
  - Calling one transaction from other
- Wallet connectivity issues
  - Account switching
- Gas fee optimization
  - Contract optimization
- User experience in blockchain transactions
  - Transaction waiting
  - Error handling

  - Status updates

- Data storage limitations
  - Documents storage

14 Future Scope

14.1 Additional Application Features

- Trusted nodes
  - Onboard car manufacturers and government agencies to integrate their data with this system.
- Purchasing with cryptocurrency
  - Add support for buying and selling using HBAR itself, or any cryptocurrency.
- Integration with Web3 Storage
  - Integration with IPFS, Pinata etc. for document tracking and uploads.
- Servicing history
  - Add support for more data points like servicing history, kilometres driven etc.

14.2 Enhanced User Features

- Role-based access control
  - Admin privileges
  - Service provider access
  - Government access
- Multi-signature transactions
  - Joint ownership
  - Corporate accounts
  - Escrow services
- Advanced search capabilities

  - Fuzzy search
  - Filter combinations
  - Saved searches

14.3 System Improvements

- Performance optimization
  - Caching mechanisms
  - Query optimization
  - Load balancing
- Enhanced security features
  - Multi-factor authentication
  - Biometric verification
  - Advanced encryption
- Mobile responsiveness
  - Progressive Web App
  - Native mobile apps
  - Offline capabilities

14.4 Integration Possibilities

- Insurance providers
  - Policy management
  - Claim processing
  - Premium calculation
- Service centers
  - Appointment scheduling
  - Service tracking
  - Parts inventory
- Government agencies

  - Tax collection
  - Regulation compliance
  - Law enforcement

15 Conclusion

The Vehicle Management System on Hedera Network provides a secure, transparent, and efficient solution for vehicle registration and management. By leveraging blockchain technology, it ensures data integrity and provides a tamper-proof record of vehicle information. The system's modular architecture and comprehensive feature set make it a robust solution for modern vehicle management needs.

Key achievements include:

- Successful implementation of decentralized vehicle registration using smart contracts
- Secure and transparent ownership transfer mechanism
- User-friendly interface for system interaction
- Robust security measures and data integrity

The system demonstrates the practical application of blockchain technology in solving real-world problems related to vehicle management and sets a foundation for future enhancements and integrations.

16 References

1. Hedera Documentation
    - Smart Contract Service
    - Consensus Service
    - Token Service
2. Solidity Documentation
    - Language Specification
    - Security Considerations
    - Best Practices

3. Angular Documentation
    - Component Architecture
    - Service Implementation
    - Routing System
4. Web3.js Documentation
    - Contract Interaction
    - Event Handling
    - Transaction Management
5. MetaMask Documentation
    - Wallet Integration
    - Transaction Signing
    - Network Management

17 Appendix

17.1 Smart Contract Code

17.1.1 VehicleLedger.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract VehicleLedger {
    struct Maintenance {
        uint date;
        string maintenanceType;
        string serviceProvider;
    }

    struct Insurance {
        string insuranceRef;
        string docHash;
        string docLink;
    }

```

```
17      struct Accident {
18          uint date;
19          string reportDocHash;
20          string reportDocLink;
21      }
22
23      struct Vehicle {
24          string regNo;
25          address currentOwner;
26          address[] pastOwners;
27          uint yearOfManufacturing;
28          Maintenance[] maintenanceHistory;
29          Insurance[] insuranceHistory;
30          Accident[] accidentHistory;
31          uint[] resellAmounts;
32      }
33
34      mapping(string => Vehicle) public vehicles;
35      address public userProfileAddress; // Address of the UserProfile contract
36
37      address public owner;
38
39      constructor() {
40          owner = msg.sender;
41      }
42
43      modifier onlyOwner() {
44          require(msg.sender == owner, "Only the creator can call this function");
45          _;
46      }
47
48      // Set the UserProfile contract address via a method.
49      function setUserProfileAddress(address _addr) public onlyOwner {
50          userProfileAddress = _addr;
51      }
52
53      // Register a new vehicle.
54      // Registration accepts only the registration number and year. History entries can be appended l
55      function registerVehicle(string calldata regNo, uint yearOfManufacturing) external {
56          // Ensure vehicle doesn't already exist.
57          require(vehicles[regNo].yearOfManufacturing == 0, "Vehicle already registered");
58
59          Vehicle storage vehicle = vehicles[regNo];
60          vehicle.regNo = regNo;
61          vehicle.currentOwner = msg.sender;
62          vehicle.yearOfManufacturing = yearOfManufacturing;
63
64          // Call UserProfile to add this vehicle to the user's profile.
```

```
        if (userProfileAddress != address(0)) {
            (bool success,) = userProfileAddress.call(
                abi.encodeWithSignature("addVehicle(address,string)", msg.sender, regNo)
            );
            require(success, "addVehicle call failed");
        }
    }

    // Append a maintenance entry. Only the current owner can update.
    function addMaintenance(string calldata regNo, uint date, string calldata maintenanceType, strin
        Vehicle storage vehicle = vehicles[regNo];
        require(vehicle.currentOwner != address(0), "Vehicle not found");
        require(vehicle.currentOwner == msg.sender, "Caller is not the owner");
        vehicle.maintenanceHistory.push(Maintenance(date, maintenanceType, serviceProvider));
    }

    // Append an insurance entry. Only the current owner can update.
    function addInsurance(string calldata regNo, string calldata insuranceRef, string calldata docHa
        Vehicle storage vehicle = vehicles[regNo];
        require(vehicle.currentOwner != address(0), "Vehicle not found");
        require(vehicle.currentOwner == msg.sender, "Caller is not the owner");
        vehicle.insuranceHistory.push(Insurance(insuranceRef, docHash, docLink));
    }

    // Append an accident entry. Only the current owner can update.
    function addAccident(string calldata regNo, uint date, string calldata reportDocHash, string cal
        Vehicle storage vehicle = vehicles[regNo];
        require(vehicle.currentOwner != address(0), "Vehicle not found");
        require(vehicle.currentOwner == msg.sender, "Caller is not the owner");
        vehicle.accidentHistory.push(Accident(date, reportDocHash, reportDocLink));
    }

    // Resale a vehicle.
    // Only the current owner can sell. Records resale amount history.
    function resaleVehicle(string calldata regNo, address newOwner, uint resellAmount) external {
        Vehicle storage vehicle = vehicles[regNo];
        require(vehicle.currentOwner == msg.sender, "Caller is not the owner");

        // Record current owner in past owners.
        vehicle.pastOwners.push(msg.sender);
        vehicle.currentOwner = newOwner;
        vehicle.resellAmounts.push(resellAmount);

        // Update UserProfile: remove vehicle from seller and add to buyer.
        if (userProfileAddress != address(0)) {
            (bool success,) = userProfileAddress.call(
                abi.encodeWithSignature("removeVehicle(address,string)", msg.sender, regNo)
            );
```

```
            require(success, "removeVehicle call failed");

            (bool successAdd,) = userProfileAddress.call(
                abi.encodeWithSignature("addVehicle(address,string)", newOwner, regNo)
            );
            require(successAdd, "addVehicle call failed");
        }
    }

    function getVehicleDetails(string calldata regNo) external view returns (
        string memory,
        address,
        address[] memory,
        uint,
        uint,
        uint,
        uint,
        uint
    ) {
        Vehicle storage vehicle = vehicles[regNo];
        require(vehicle.yearOfManufacturing != 0, "Vehicle not found");
        return (
            vehicle.regNo,
            vehicle.currentOwner,
            vehicle.pastOwners,
            vehicle.yearOfManufacturing,
            vehicle.maintenanceHistory.length,
            vehicle.insuranceHistory.length,
            vehicle.accidentHistory.length,
            vehicle.resellAmounts.length
        );
    }

    function getMaintenanceHistory(string calldata regNo) external view returns (Maintenance[] memor
        return vehicles[regNo].maintenanceHistory;
    }

    function getInsuranceHistory(string calldata regNo) external view returns (Insurance[] memory) {
        return vehicles[regNo].insuranceHistory;
    }

    function getAccidentHistory(string calldata regNo) external view returns (Accident[] memory) {
        return vehicles[regNo].accidentHistory;
    }

    function getResellHistory(string calldata regNo) external view returns (uint[] memory) {
        return vehicles[regNo].resellAmounts;
    }
```

```
161
162     function getPastOwnerHistory(string calldata regNo) external view returns (address[] memory) {
163         return vehicles[regNo].pastOwners;
164     }
165 }
```

17.1.2 UserProfile.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  contract UserProfile {
5      struct User {
6          address walletAddress;
7          string name;
8          string phone;
9          string[] vehicles; // vehicle registration numbers owned by user
10     }
11
12     mapping(address => User) public users;
13
14     // Internal function to add a user with no details if not present.
15     function ensureUserExists(address userAddr) internal {
16         if (users[userAddr].walletAddress == address(0)) {
17             // Create a new user with default empty details.
18             users[userAddr] = User({
19                 walletAddress: userAddr,
20                 name: "",
21                 phone: "",
22                 vehicles: new string[](0)
23             });
24         }
25     }
26
27     // Create or update user profile.
28     // Called by the user themselves.
29     function updateUserProfile(string calldata _name, string calldata _phone) external {
30         ensureUserExists(msg.sender);
31         User storage user = users[msg.sender];
32         user.name = _name;
33         user.phone = _phone;
34     }
35
36     // Append a vehicle registration number to user's profile.
37     // Called from from the VehicleLedger contract.
38     function addVehicle(address userAddr, string calldata regNo) external {
39         ensureUserExists(userAddr);
```

```
40          users[userAddr].vehicles.push(regNo);
41      }
42
43      // Remove a vehicle registration number from user's profile.
44      // Called from the VehicleLedger contract.
45      function removeVehicle(address userAddr, string calldata regNo) external {
46          User storage user = users[userAddr];
47          uint len = user.vehicles.length;
48          for (uint i = 0; i < len; i++) {
49              if (keccak256(bytes(user.vehicles[i])) == keccak256(bytes(regNo))) {
50                  // Remove element by swapping with last and popping the array.
51                  user.vehicles[i] = user.vehicles[len - 1];
52                  user.vehicles.pop();
53                  break;
54              }
55          }
56      }
57
58      // Retrieve user profile details.
59      // Note: This functions is not marked as view, because they may modify state by ensuring the use
60      function getUserProfile(address userAddr) external returns (string memory, string memory, string
61          ensureUserExists(userAddr);
62          User storage user = users[userAddr];
63          return (user.name, user.phone, user.vehicles);
64      }
65
66      // Getter for the wallet address associated with the user profile.
67      // Note: This functions is not marked as view, because they may modify state by ensuring the use
68      function getUserWallet(address userAddr) external returns (address) {
69          ensureUserExists(userAddr);
70          return users[userAddr].walletAddress;
71      }
72
73      // Getter for the user's name.
74      function getUserName(address userAddr) external view returns (string memory) {
75          return users[userAddr].name;
76      }
77
78      // Getter for the user's phone number.
79      function getUserPhone(address userAddr) external view returns (string memory) {
80          return users[userAddr].phone;
81      }
82
83      // Getter for the array of vehicle registration numbers.
84      function getUserVehicles(address userAddr) external view returns (string[] memory) {
85          return users[userAddr].vehicles;
86      }
87  }
```

17.2 Deployment Guide

1. Prerequisites
    - Node.js 20.11.0 or later
    - Angular CLI 19.2.0
    - MetaMask wallet
    - Hedera testnet account
2. Setup Steps
    - Clone repository
    - Install dependencies
    - Configure environment variables
    - Deploy smart contracts
    - Start frontend application
3. Testing
    - Run unit tests
    - Execute integration tests
    - Perform security audits

18 Deployment Information

The application is deployed with the following details:

18.1 Project Repository

- GitHub: https://github.com/dharmendra912/VehicleResellingDapp

18.2 Smart Contracts

- VehicleLedger.sol
    - Contract ID: 0.0.5864405

  - EVM Address: 0x000000000000000000000000000000000000597bd5
  - Explorer: Contract Explorer

- UserProfile.sol
  - Contract ID: 0.0.5864437
  - EVM Address: 0x000000000000000000000000000000000000597bf5
  - Explorer: Contract Explorer

18.3 IPFS Gateway

The frontend is deployed on IPFS with the following gateway:

- IPFS Gateway