GOVERNMENT COLLEGE OF ENGINEERING,

ERODE: 638 316

(Formerly known as Institute of Road and Transport Technology)

RECORD NOTE BOOK

Reg. No._____

Certified that this is the Bonafide record of work done by Selvan / Selvi _____ of the **FIFTH** Semester of **B. TECH INFORMATION TECHNOLOGY** during the Academic year **2024 – 2025** in the CCS339 CRYPTOCURRENCY AND BLOCKCHAIN TECHNOLOGIES.

Staff-in-charge **Head of the Department**

Submitted for the University Practical Examination on _____at the Government College of Engineering, Erode.

Date: _____

Internal Examiner **External Examiner**

INDEX

EX. NO: 01	INSTALL AND UNDERSTAND DOCKER CONTAINER, NODE.JS, JAVA AND HYPERLEDGER FABRIC, ETHEREUM AND PERFORM NECESSARY SOFTWARE INSTALLATION ON LOCAL MACHINE/CREATE INSTANCE ON CLOUD TO RUN.
DATE:	

AIM

To install and understand the Docker container, Node.js, Java, Hyperledger Fabric, and Ethereum by performing the necessary software installations on a Windows machine.

UNDERSTANDING DOCKER

Docker is a containerization platform which is used for packaging an application and its dependencies together within a Docker container. This ensures the effortless and smooth functioning of our application irrespective of the changes in the environment.

Talking about Docker Container it is nothing but a standardized unit that is used to deploy a particular application or environment and can be built dynamically. We can have any container such as Ubuntu, CentOS, etc. based on your requirement with respect to Operating Systems.

Docker Container: A standardized unit used to deploy a specific application or environment. Containers can be built dynamically based on requirements, using different operating systems like Ubuntu, CentOS, etc.

Docker File: A text document that contains a list of commands that can be invoked to assemble an image. Docker automatically builds images by reading instructions from the Dockerfile.

Docker Image: A template used to build Docker containers. These read-only templates serve as the building blocks of a Docker container.

Docker Registry: A storage location for Docker images. It can be a local repository or a public repository like Docker Hub, allowing multiple users to collaborate on building an application.

Docker Container: The running instance of a Docker image that holds the complete package required to execute the application.

Basic Commands

Command	Usage
docker run <image>	To execute an image and build a container
docker ps	List running containers

Command	Usage
docker stop <container_id>	List all containers (including stopped ones)
docker ps -a	List all containers (including stopped ones)
docker rm <container_id>	Remove a container
docker rmi <image_id>	Remove an image
docker images	View Docker images
docker logs <container_id>	View logs of a container

Installation Docker

Step 1: Download Docker Desktop.

1. Go to the Docker Desktop for Windows website. (https://www.docker.com/products/docker-desktop/)
2. Click the **Get Docker** button to download the installer for windows.

Step 2: Install Docker Desktop.

1. Run the Installer:
    - Locate the downloaded Docker Desktop installer (usually in the Downloads folder).
    - Double-click the installer to run it.
2. Installation Wizard:
    - Follow the on-screen instructions in the installation wizard.
    - During installation, you may see options for "Install required Windows components for WSL 2." Ensure this option is checked to enable Windows Subsystem for Linux (WSL).
3. Enable WSL 2 (if prompted):
    - If you don't have WSL 2 enabled, the installer will guide you through enabling it. You may need to restart your computer after enabling WSL.

4. Configuration:

    - After installation, launch Docker Desktop from the Start menu.
    - Docker may prompt you to log in or create a Docker Hub account. You can skip this step if you do not wish to sign in.

Step 3: Start Docker Desktop

1. Once installed, Docker Desktop should start automatically. If it doesn't, you can manually launch it from the Start menu.
2. You will see a whale icon in the system tray indicating that Docker is running.

Step 4: Verify Docker Installation

1. Open Command Prompt, Press Win + R, type cmd, and press Enter.

    1. Check Docker Version:

        Run the following command in Command Prompt

        >docker –-version

        Docker version 27.1.1, build 6312585

        To ensure everything is working correctly, run the following command:

        >docker run hello-world

        Unable to find image 'hello-world:latest' locally
        latest: Pulling from library/hello-world
        c1ec31eb5944: Pull complete

        Digest: sha256:
        d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348

        Status: Downloaded newer image for hello-world:latest

        Hello from Docker!
        This message shows that your installation appears to be working correctly.

UNDERSTANDING NODE.JS

Node.js is an open-source, cross-platform JavaScript runtime environment that executes JavaScript code outside a web browser. It is popular for its non-blocking, event-driven architecture, which makes it efficient and suitable for building scalable network applications.

Node.js uses the V8 JavaScript engine to execute code and the libuv library to handle asynchronous operations. This combination allows Node.js to perform non-blocking I/O operations, making it highly performant. Common use cases for Node.js include web servers, real-time applications (like chat apps), REST APIs, command-line tools, and microservices.

Installing node.js on Windows

Step 1: Download the Installer

1. Go to the official Node.js website (https://nodejs.org/en). Click on the LTS (Long Term Support) version recommended for most users. This will download the installer file (.msi).

Step 2: Run the Installer

1. Locate the downloaded .msi file and double-click to run it. This will open the Node.js Setup Wizard.

Step 3: Accept the License Agreement

1. Read the license agreement, check the box to accept the terms, and click "Next".

Step 4: Choose Installation Location

1. Select the destination folder where you want to install Node.js. The default location is usually fine. Click "Next".

Step 5: Complete the Installation

1. Once the installation is complete, click "Finish" to exit the Setup Wizard.

Step 6: Verify installation

1. Open a command prompt (press Win + R, type cmd, and press Enter). Verify the installation by running the following commands:
    1. node -v (this should display the Node.js version)
    2. npm -v (this should display the npm version)

Basic commands

Command	Usage
npm init	Initializes a new Node.js project, creating a package.json file.
npm install <pkg_name>	Installs a package locally in your project
npm install	Installs all dependencies listed in package.json.
npm list	Lists all installed packages in the current project
npm run <script_name>	Runs a script defined in the scripts
npm uninstall <pkg_name>	Uninstalls a package from your project
npm cache clean –force	Cleans the npm cache to resolve issues

DOCKERIZING THE SIMPLE NODE.JS APPLICATION

Step 1: Create a Simple Node.js Application

1. **Initialize your Node.js project:**

Open your terminal and create a new directory for your project and install necessary packages:

```
>mkdir my-node-app
>cd my-node-app
>npm init -y
>npm install express
```

2. **Create an index.js file and add the following code:**

```
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
  res.send('Hello, World!');
});
app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`);
});
```

Step 2: Create a Dockerfile

1. **Create a Dockerfile in your project directory add the following content:**

```
# Use the official Node.js image as the base image
FROM node:14

# Create and set the working directory
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install the app dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Expose the port the app runs on
EXPOSE 3000

# Define the command to run the app
CMD ["node", "index.js"]
```

Step 3: Create a .dockerignore File

1. Create a .dockerignore file in your project directory and add the following content:

```
node_modules
npm-debug.log
```

Step 4: Build the Docker Image

1. Open your terminal and navigate to your project directory.

2. Build the Docker image using the following command:

```
>docker build -t my-node-app
```

Step 5: Run the Docker Container

1. Run the Docker container using the following command:

   >docker run -p 3000:3000 my-node-app

2. Open your web browser and navigate to http://localhost:3000,You should see the message "Hello, World!".

Step 6: Verify the Docker Container

1. List the running containers to verify your container is running:

   >docker ps

2. Stop the container if needed:

   >docker stop <container_id>

INSTALLING JAVA

Step 1: Download Java

- Go to the Oracle JDK download page (or the version you need).
- Accept the license agreement and download the Windows Installer.

Step 2: Install Java

- Run the downloaded installer.
- Follow the installation instructions and choose the default settings.

Step 3: Set JAVA_HOME

- Right-click on This PC or My Computer and select Properties.
- Click on Advanced system settings.
- In the System Properties window, click on the Environment Variables button.
- In the Environment Variables window, click on New under System variables.
  - Variable name: JAVA_HOME
  - Variable value: Path to your Java installation
  - (e.g., C:\Program Files\Java\jdk-11.x.x).
- Click OK to save the new variable.

Step 4: Update PATH

- In the same Environment Variables window, find and select the Path variable under System variables, then click Edit.
- Click New and add: %JAVA_HOME%\bin
- Click OK to save and close all dialog boxes.

Step 5: Verify Installation

Open Command Prompt and run:

java -version

UNDERSTANDING HYPERLEDGER FABRICS

Hyperledger Fabric is an open-source framework for building blockchain-based applications and networks. It is designed for enterprise solutions and provides a modular architecture that allows organizations to create and manage their own distributed ledgers with specific requirements. Here are some key features and concepts of Hyperledger Fabric:

1. **Modular Architecture**: Hyperledger Fabric has a pluggable architecture that enables developers to customize various components like consensus algorithms, membership services, and data storage.
2. **Channels**: Channels are private subnets within a blockchain network that allow a group of participants to transact in a secure environment, isolating their data from other network participants.
3. **Smart Contracts (Chain code)**: Hyperledger Fabric allows developers to write business logic in the form of smart contracts, known as chain code. These contracts execute on the blockchain and enforce the rules of transactions.
4. **Permissioned Network**: Hyperledger Fabric operates in a permissioned environment, meaning that participants are known and authorized to join the network. This feature enhances privacy and control over data access.
5. **Endorsement Policies**: Endorsement policies define the conditions under which transactions can be considered valid. A transaction must be endorsed by a specific number of peers before it is committed to the ledger.

Prerequisites for installing Hyperledger fabric

Step 1: Install Docker

1. Download Docker Desktop for Windows from Docker's official site.
2. Follow the installation instructions and ensure Docker is running.

Step 2: Install WSL (Windows Subsystem for Linux):

1. Open PowerShell as an Administrator and run the following command:

   >wsl –install

2. This will install WSL and the default Ubuntu distribution. You can also install other distributions from the Microsoft Store if preferred.

Step 3: Install Git:

1. Download and install Git from Git's official site.
2. During installation, select the default options.

Step 4: Install Node.js:

1. Download the latest LTS version of Node.js from Node.js official site.
2. Follow the installation instructions.

Step-by-Step Installation of Hyperledger Fabric

1. **Open WSL**:

   - Open your installed WSL terminal (e.g., Ubuntu).

2. **Install Required Packages**:

   $sudo apt-get update
   $sudo apt-get install -y curl wget apt-transport-https

3. **Install Docker and Docker Compose**:

   - You should already have Docker Desktop installed. Ensure it is running.
   - For Docker Compose, use:

     $sudo apt-get install -y docker-compose

4. **Download Hyperledger Fabric Samples, Binaries, and Docker Images**:

   $mkdir -p ~/fabric-samples
   $cd ~/fabric-samples
   $curl -sSL https://bit.ly/2ysbOFE | bash -s

5. **Set Environment Variables**:

    - Add the Fabric binaries to your path:

        $****echo 'export PATH=****$*PATH:****$****HOME/fabric-samples/bin' >> ~/.bashrc*
        $*source ~/.bashrc*

6. **Start the Sample Network**:

    - Navigate to one of the sample directories (e.g., first-network):

        $*cd ~/fabric-samples/first-network*

    - Start the network:

        $*./byfn.sh up*

    - This command will start the Docker containers for the Fabric network.

7. **Verify Installation**:

    - Check the logs to confirm that the network started successfully:

        $*docker logs -f peer0.org1.example.com*

8. **Interacting with the Network**:

    - You can now use the Fabric SDK or command-line tools to interact with your blockchain network.

UNDERSTANDING ETHEREUM

Ethereum is a decentralized, open-source blockchain platform that enables developers to build and deploy smart contracts and decentralized applications (dApps). Launched in 2015 by Vitalik Buterin and a team of developers, Ethereum extends the capabilities of blockchain technology beyond simple transactions by allowing programmable agreements that execute automatically when predefined conditions are met.At its core, Ethereum features its native cryptocurrency, Ether (ETH), which is used to facilitate transactions, pay for computational services, and incentivize network participants.

The Ethereum Virtual Machine (EVM) provides a runtime environment for executing smart contracts, ensuring consistency and security across the network.

Installation Steps

1. **Open Command Prompt**:
    - Press Windows + R, type cmd, and hit Enter to open the Command Prompt.
2. **Install Windows Subsystem for Linux (WSL)**:
    - In the Command Prompt, type the following command and press Enter:

        >wsl –install

    - This installs WSL and the default Ubuntu distribution. Restart your computer if prompted.
3. **Open WSL (Ubuntu)**:
    - After installation, search for **Ubuntu** in the Start menu and open it.
4. **Update Package List**:
    - In the Ubuntu terminal, run:

        $*sudo apt update*

    - This command updates the list of available packages.
5. **Install Geth (Go Ethereum)**:
    - In the Ubuntu terminal, run the following commands to install Geth:

        $*sudo add-apt-repository ppa:ethereum/ethereum*
        $*sudo apt update*
        $*sudo apt install geth*

    - These commands add the Ethereum PPA (Personal Package Archive) and install Geth.

6. **Install Hardhat** (for smart contract development):

    - Create a new project directory:

        >mkdir ~/eth-project
        >cd ~/eth-project

    - Initialize a new Node.js project:

        >npm init -y

    - Install Hardhat:

        >npm install --save-dev hardhat

7. **Set Up a Hardhat Project**:

    - Run the Hardhat initialization command:

        >npx hardhat

    - Choose **Create a sample project** and follow the prompts. This sets up a sample Ethereum project.

8. **Install MetaMask**:

    - Open your web browser (Chrome, Firefox, etc.).
    - Go to the MetaMask website and install the MetaMask extension.
    - Follow the instructions to set up your wallet (create a new wallet or import an existing one).

9. **Start Geth**:

    - Back in the WSL terminal, start Geth to connect to the Ethereum network:

        >geth --syncmode "fast" --http --http.port 8545 --http.api "web3,eth,personal" --allow-insecure-unlock

    - This command starts Geth and allows it to sync with the Ethereum blockchain.

10. **Verify Geth Installation**:

- Open another terminal (you can use the same WSL terminal) and run:

  >geth attach

- This command connects you to the running Geth instance.

11. **Using Hardhat**:

- You can now use Hardhat commands to compile and deploy smart contracts. For example, to compile your contracts, use:

  >npx hardhat compile

RESULT

Thus, the Installation of Docker Container, Node.js and HyperledgerFabric and Etherum Network are executed successfully.

EX.NO: 2	CREATE AND DEPLOY A BLOCKCHAIN NETWORK
DATE:	

AIM:

To create and deploy a blockchain network to understand its foundational mechanisms and applications in secure, decentralized data management.

PROCEDURE

1. Prerequisites

1. Install MetaMask browser extension (for interacting with the Ethereum network).

2. Create an Ethereum account in MetaMask.

3. Fund your account with Ether (for deploying contracts).

4. Install Remix IDE (web-based or desktop application).

2. Setting up the environment

Clone the Repository:

>git clone https://github.com/IT-Blockchain-Exp/crowdFundingApp.git

//(packages details)

>cd crowdfundingApp

Section 1: Setting up Remix IDE

1. Open Remix IDE: Go to (link unavailable) or open the desktop application.

2. Create a new project: Click on "Create New Project" and choose a project name.

3. Create a new file: Click on "File" > "New File" and name it (e.g., "MyContract.sol").

Section 2: Writing and Compiling Solidity Code

1. Write your Solidity code: Copy and paste your contract code into the "MyContract.sol" file.

2. Compile the code: Click on the "Compile" button (or press Ctrl+S) to compile the code.

Section 3: Deploying the Contract

1. Open the "Deploy" tab: Click on the "Deploy" tab in the Remix IDE.

2. Select environment: Choose "Injected Web3" (connects to MetaMask).

3. Select account: Choose your MetaMask account.

4. Set gas limit: Adjust the gas limit according to your contract's requirements.

5. Deploy contract: Click on "Deploy" to deploy the contract.

Section 4: Getting the Contract Address

1. Wait for deployment: Wait for the contract to deploy (may take a few seconds).

2. Check transaction hash: Verify the transaction hash in the "Deploy" tab.

3. Get contract address: Click on the "Contract" tab and find the contract address.

4. Copy contract address: Copy the contract address for future reference.

CODE: Crowdfunding.sol

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

contract CrowdFunding {
    struct Campaign {
        address owner;
        string title;
        string description;
        uint256 target;
        uint256 deadline;
        uint256 amountCollected;
        string image;
        address[] donators;
        uint256[] donations;
        uint256[] donationTimestamps; // Store timestamps of donations
    }

    mapping(uint256 => Campaign) public campaigns;
    uint256 public numberOfCampaigns = 0;

    event CampaignCreated(uint256 campaignId, address indexed owner, uint256 target, uint256 deadline);
    event DonationReceived(uint256 campaignId, address indexed donator, uint256 amount, uint256 timestamp);
    event Withdrawn(uint256 campaignId, address indexed owner, uint256 amount);
    event Refunded(uint256 campaignId, address indexed donator, uint256 amount);

    function createCampaign(
        string memory _title,
```

```
        string memory _description,
        uint256 _target,
        uint256 _deadline,
        string memory _image
    ) public returns (uint256) {
        require(_target > 0, "Target must be greater than zero.");
        require(_deadline > block.timestamp, "Deadline should be in the future.");

        Campaign storage campaign = campaigns[numberOfCampaigns];
        campaign.owner = msg.sender;
        campaign.title = _title;
        campaign.description = _description;
        campaign.target = _target;
        campaign.deadline = _deadline;
        campaign.amountCollected = 0;
        campaign.image = _image;

        emit CampaignCreated(numberOfCampaigns, msg.sender, _target, _deadline);
        numberOfCampaigns++;
        return numberOfCampaigns - 1;
    }

    function donateToCampaign(uint256 _id) public payable {
        Campaign storage campaign = campaigns[_id];
        require(block.timestamp < campaign.deadline, "Campaign deadline has passed.");
        require(msg.value > 0, "Donation must be greater than zero.");

        campaign.donators.push(msg.sender);
        campaign.donations.push(msg.value);
        campaign.donationTimestamps.push(block.timestamp); // Store timestamp
        campaign.amountCollected += msg.value;

        emit DonationReceived(_id, msg.sender, msg.value, block.timestamp);
    }

    function withdraw(uint256 _id) public {
        Campaign storage campaign = campaigns[_id];
        require(msg.sender == campaign.owner, "Only the campaign owner can withdraw
funds.");
        require(block.timestamp >= campaign.deadline, "Cannot withdraw before the
deadline.");
        require(campaign.amountCollected >= campaign.target, "Campaign not fully
funded.");

        uint256 amount = campaign.amountCollected;
```

```
        campaign.amountCollected = 0;

        (bool sent, ) = payable(campaign.owner).call{value: amount}("");
        require(sent, "Failed to send funds.");

        emit Withdrawn(_id, campaign.owner, amount);
    }

    // Function to handle automatic refunds
    function checkRefunds(uint256 _id) public {
        Campaign storage campaign = campaigns[_id];
        require(block.timestamp >= campaign.deadline, "Campaign is still active.");
        require(campaign.amountCollected < campaign.target, "Campaign was successful, no
refunds allowed.");

        for (uint256 i = 0; i < campaign.donators.length; i++) {
            address donator = campaign.donators[i];
            uint256 donationAmount = campaign.donations[i];

            // Refund the donation
            (bool sent, ) = payable(donator).call{value: donationAmount}("");
            require(sent, "Refund failed.");

            emit Refunded(_id, donator, donationAmount);
        }

        // Reset the campaign to avoid re-refunds
        campaign.amountCollected = 0;
        delete campaign.donators;
        delete campaign.donations;
        delete campaign.donationTimestamps;
    }

    function getDonators(uint256 _id) public view returns (
        address[] memory,
        uint256[] memory,
        uint256[] memory
    ) {
        return (campaigns[_id].donators, campaigns[_id].donations,
campaigns[_id].donationTimestamps);
    }

    function getCampaigns() public view returns (Campaign[] memory) {
        Campaign[] memory allCampaigns = new Campaign[](numberOfCampaigns);
        for (uint256 i = 0; i < numberOfCampaigns; i++) {
```

```
        allCampaigns[i] = campaigns[i];
      }
      return allCampaigns;
    }
}
```

APP.JS:

```
import React, { useEffect, useState, useCallback } from 'react';
import './App.css';
import { ethers } from 'ethers';
import CrowdFunding from './CrowdFunding.json';

function App() {
  const [campaigns, setCampaigns] = useState([]);
  const [donationAmount, setDonationAmount] = useState({});
  const [donationRefundAddress, setDonationRefundAddress] = useState({});
  const contractAddress = "0x9901b317028800297fd21e51ad6bb9bd2c5d33f2"; // Replace
with your contract address

  const fetchCampaigns = useCallback(async () => {
    try {
      const { ethereum } = window;
      if (!ethereum) {
        alert("Please install MetaMask!");
        return;
      }
      const provider = new ethers.providers.Web3Provider(ethereum);
      const signer = provider.getSigner();
      const crowdFundingContract = new ethers.Contract(contractAddress,
CrowdFunding.abi, signer);
      const allCampaigns = await crowdFundingContract.getCampaigns();
      setCampaigns(allCampaigns);
    } catch (error) {
      console.error("Error fetching campaigns:", error);
      alert("Failed to fetch campaigns.");
    }
  }, [contractAddress]);

  useEffect(() => {
    fetchCampaigns();
  }, [fetchCampaigns]);

  const handleCreateCampaign = async (event) => {
```

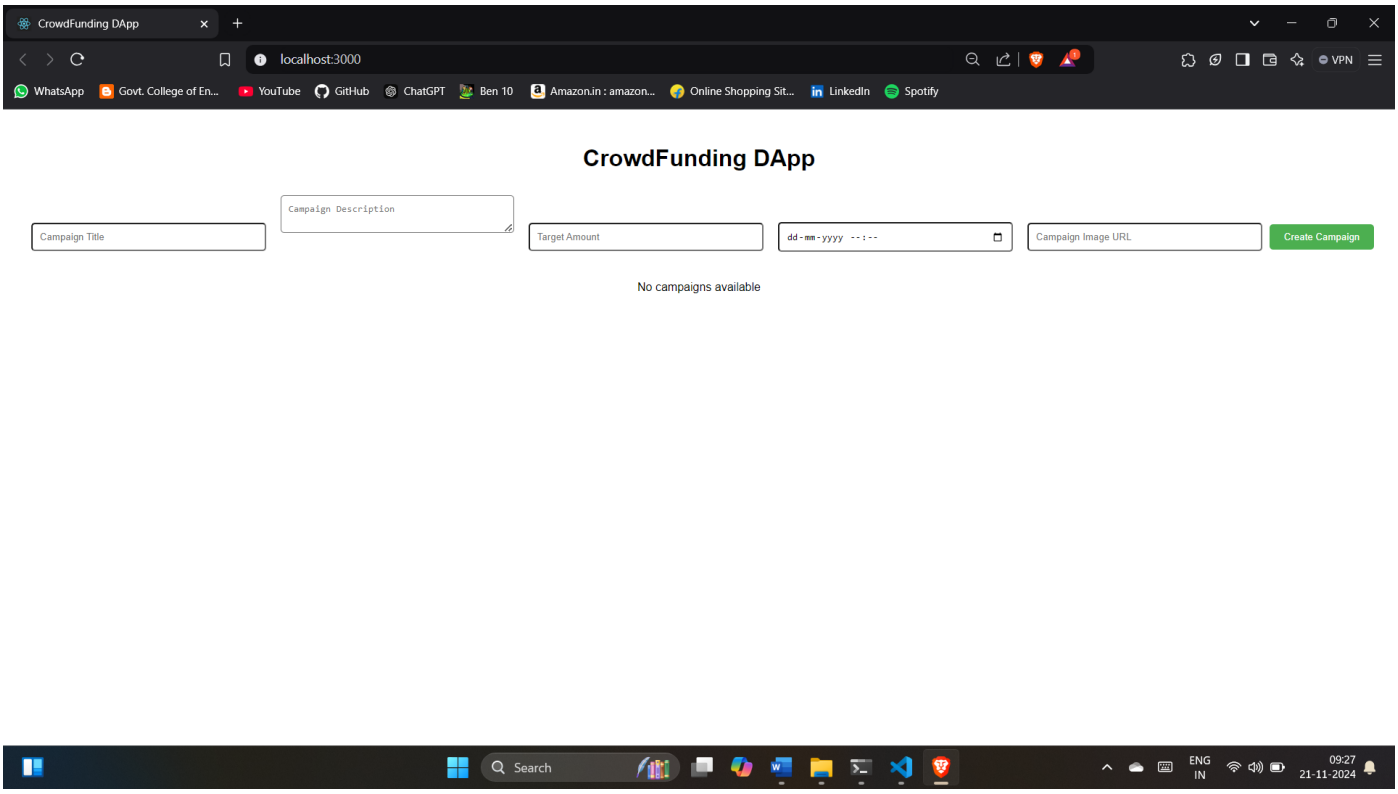
```
    event.preventDefault();
    const title = event.target.title.value;
    const description = event.target.description.value;
    const target = ethers.utils.parseEther(event.target.target.value);
    const deadline = Math.floor(new Date(event.target.deadline.value).getTime() / 1000);
    const image = event.target.image.value;

    try {
      const { ethereum } = window;
      if (!ethereum) {
        alert("Please install MetaMask!");
        return;
      }
      const provider = new ethers.providers.Web3Provider(ethereum);
      const signer = provider.getSigner();
      const crowdFundingContract = new ethers.Contract(contractAddress,
CrowdFunding.abi, signer);
      const transaction = await crowdFundingContract.createCampaign(
        title,
        description,
        target,
        deadline,
        image,
        { gasLimit: 3000000 }
      );

      await transaction.wait();
      fetchCampaigns();
      alert("Campaign created successfully!");
    } catch (error) {
      console.error("Error creating campaign:", error);
      alert("Failed to create campaign.");
    }
  };

  const handleDonate = async (campaignIndex) => {
    const amount = donationAmount[campaignIndex];
    const refundAddress = donationRefundAddress[campaignIndex];
    if (!amount || isNaN(amount) || parseFloat(amount) <= 0) {
      alert("Please enter a valid donation amount.");
      return;
    }
    if (!refundAddress) {
      alert("Please enter a valid refund address.");
      return;
```

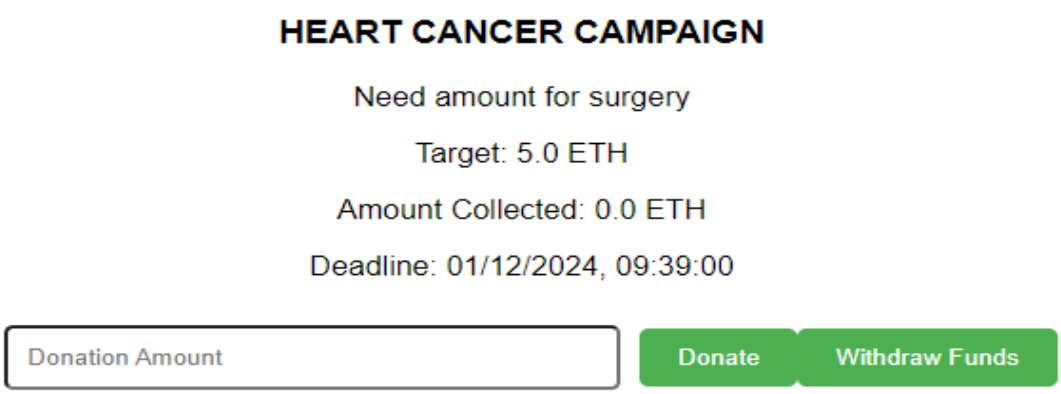
```
    }

    try {
      const { ethereum } = window;
      if (!ethereum) {
        alert("Please install MetaMask!");
        return;
      }
      const provider = new ethers.providers.Web3Provider(ethereum);
      const signer = provider.getSigner();
      const crowdFundingContract = new ethers.Contract(contractAddress,
CrowdFunding.abi, signer);
      const transaction = await crowdFundingContract.donateToCampaign(campaignIndex,
refundAddress, {
        value: ethers.utils.parseEther(amount.toString()),
        gasLimit: 3000000
      });

      await transaction.wait();
      fetchCampaigns();
      alert("Donation successful!");
    } catch (error) {
      console.error("Error donating:", error);
      alert("Failed to donate.");
    }
  };

  const handleCheckRefunds = async (campaignIndex) => {
    try {
      const { ethereum } = window;
      if (!ethereum) {
        alert("Please install MetaMask!");
        return;
      }
      const provider = new ethers.providers.Web3Provider(ethereum);
      const signer = provider.getSigner();
      const crowdFundingContract = new ethers.Contract(contractAddress,
CrowdFunding.abi, signer);
      const transaction = await crowdFundingContract.checkRefunds(campaignIndex, {
gasLimit: 3000000 });

      await transaction.wait();
      fetchCampaigns();
      alert("Refunds processed successfully for campaign " + campaignIndex);
    } catch (error) {
```

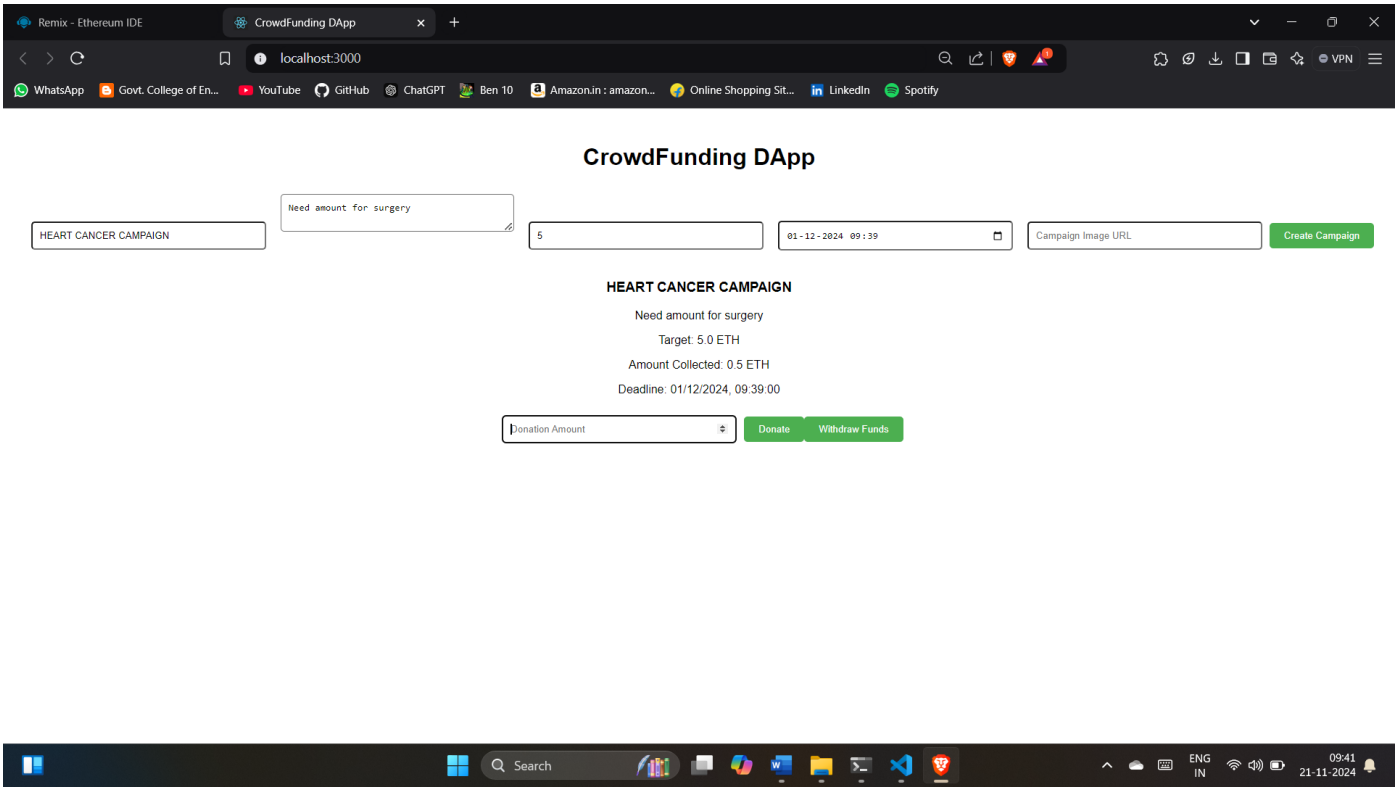
```
    console.error("Error processing refunds:", error);
    alert("Failed to process refunds.");
   }
  };

  return (
   <div className="App">
    <h1>Crowdfunding DApp</h1>
    <form onSubmit={handleCreateCampaign}>
     <input type="text" name="title" placeholder="Campaign Title" required />
     <textarea name="description" placeholder="Description" required></textarea>
     <input type="number" name="target" placeholder="Target Amount (ETH)" required
/>
     <input type="datetime-local" name="deadline" required />
     <input type="text" name="image" placeholder="Image URL" required />
     <button type="submit">Create Campaign</button>
    </form>
    <div>
     <h2>Active Campaigns</h2>
     {campaigns.length === 0 ? (
      <p>No campaigns available.</p>
     ) : (
      campaigns.map((campaign, index) => (
       <div key={index}>
        <h3>{campaign.title}</h3>
        <p>{campaign.description}</p>
        <p>Target: {ethers.utils.formatEther(campaign.target)} ETH</p>
        <p>Collected: {ethers.utils.formatEther(campaign.amountCollected)} ETH</p>
        <p>Deadline: {new Date(campaign.deadline * 1000).toLocaleString()}</p>
        <input
         type="number"
         placeholder="Donation Amount (ETH)"
         onChange={(e) => setDonationAmount((prev) => ({ ...prev, [index]:
e.target.value }))}
        />
        <input
         type="text"
         placeholder="Your Refund Address"
         onChange={(e) => setDonationRefundAddress((prev) => ({ ...prev, [index]:
e.target.value }))}
        />
        <button onClick={() => handleDonate(index)}>Donate</button>
        {Date.now() / 1000 > campaign.deadline && campaign.amountCollected <
campaign.target && (
         <button onClick={() => handleCheckRefunds(index)}>Check Refunds</button>
```

```
            )}
          </div>
        ))
      )}
    </div>
  </div>
 );
}

export default App;
```

OUTPUT:







1. Campaign Creation:
    a. A campaign is created successfully.
2. Starting the Donation:
    a. The donation process is initiated.
3. Donation Process:
    a. Donations are made using the MetaMask wallet.
4. Front-end Update:
    a. Upon successful donation, the front-end reflects the updated donation status.

RESULT:

Thus a crowd funding app has been developed successfully using blockchain technology.

EX. NO: 03	To develop an application that interacts with a blockchain network, executing transactions and testing its rules.
DATE:	

AIM

The aim of this experiment is to create and deploy a secure voting application on the Ethereum Blockchain Network that ensures the transparency and immutability of votes, while enabling interaction with the blockchain to execute transactions and test network rules.

PROCEDURE

1. Prerequisites

Before starting, ensure you have the following installed:

1. Node.js and npm
2. Hardhat
3. MetaMask
4. Git

2. Setting Up the Development Environment

a. Clone the Repository:

> git clone https://github.com/IT-Blockchain-Exp/votingApp.git (packages details)

> cd votingApp

b. Install Dependencies and Set Up Hardhat:

> npm install

> npx hardhat

3. Smart Contract Development

a. Create the Smart Contract

Navigate to the contracts directory and create a new file named Voting.sol. Write the smart contract code to handle voting-process.

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

contract Voting {
struct Candidate {
string name;
uint256 voteCount;
}

Candidate[] public candidates;
address owner;
mapping(address => bool) public voters;

uint256 public votingStart;
uint256 public votingEnd;

constructor(string[] memory _candidateNames, uint256 _durationInMinutes) {
for (uint256 i = 0; i < _candidateNames.length; i++) {
candidates.push(Candidate({
name: _candidateNames[i],
voteCount: 0
}));
}
owner = msg.sender;
votingStart = block.timestamp;
votingEnd = block.timestamp + (_durationInMinutes * 1 minutes);
}
modifier onlyOwner {
require(msg.sender == owner);
_;
}
function addCandidate(string memory _name) public onlyOwner {
candidates.push(Candidate({
name: _name,
voteCount: 0
}));
}
function vote(uint256 _candidateIndex) public {
require(!voters[msg.sender], "You have already voted.");
require(_candidateIndex < candidates.length, "Invalid candidate index.");
candidates[_candidateIndex].voteCount++;
```

```
voters[msg.sender] = true;
}

function getAllVotesOfCandiates() public view returns (Candidate[] memory){
return candidates;
}

function getVotingStatus() public view returns (bool) {
return (block.timestamp >= votingStart && block.timestamp < votingEnd);
}

function getRemainingTime() public view returns (uint256) {
require(block.timestamp >= votingStart, "Voting has not started yet.");
if (block.timestamp >= votingEnd) {
return 0;
}
return votingEnd - block.timestamp;
}

function getWinner() public view returns (string memory winnerName, uint256
winnerVoteCount) {
require(block.timestamp >= votingEnd, "Voting has not ended yet.");

uint256 highestVoteCount = 0;
uint256 winnerIndex = 0;

for (uint256 i = 0; i < candidates.length; i++) {
if (candidates[i].voteCount > highestVoteCount) {
highestVoteCount = candidates[i].voteCount;
winnerIndex = i;
}
}
winnerName = candidates[winnerIndex].name;
winnerVoteCount = candidates[winnerIndex].voteCount;
}
}
```

b. Compile the smart contract

```
> npx hardhat compile
```

4. Deploying the Smart Contract

a. Create a Deployment Script

```
async function main() {
const Voting = await ethers.getContractFactory("Voting");

// Start deployment, returning a promise that resolves to a contract object
const Voting_ = await Voting.deploy(["Mark", "Mike", "Henry", "Rock"], 90);
console.log("Contract address:", Voting_.address);
}

main()
.then(() => process.exit(0))
.catch((error) => {
console.error(error);
process.exit(1);
});
```

b. Deploy the Contract to the Volta Network

Modify the hardhat.config.js to include configurations for the Volta networ.

```
require("@nomiclabs/hardhat-waffle");
require("@nomiclabs/hardhat-ethers");
const API_URL = "https://YOUR_NETWORK_RPC_URL"; // Replace with
your RPC URL
const PRIVATE_KEY = "YOUR-PRIVATE-KEY";

module.exports = {
   solidity:  "0.8.0",
   networks: {
      YOUR_NETWORK: {
         url: API_URL,
         accounts: [`0x${PRIVATE_KEY}`]
      }
   }
};
```

c. Deploy the smart contract to the Volta network

```
>npx hardhat run scripts/deploy.js --network volta
```

5. Setting Up the Blockchain Network

a. Connect MetaMask to Volta Network

Open MetaMask and add a new network with the following settings:

i. Network Name : Volta
ii. New RPC URL : https://volta-rpc.energyweb.org
iii. Chain ID : 73799
iv. Currency Symbol : volta

b. Import Accounts to MetaMask

Import the accounts used in your Hardhat configuration by using the private keys.

6. Developing the Frontend

a. Setting Up React App

Navigate to the client directory and install necessary dependencies and start the React development server.

>cd client
>npm install
>npm start

7. Connecting to the Smart Contract

Use the ethers.js library to connect to the Ethereum network and interact with the deployed smart contract. Update the frontend components to include logic for voting.

```
import { useState, useEffect } from "react";
import { ethers } from "ethers";
import { contractAbi, contractAddress } from "./Constant/constant";
import Login from "./Components/Login";
import Finished from "./Components/Finished";
import Connected from "./Components/Connected";
import "./App.css";
function App() {
```

```
const [provider, setProvider] = useState(null);
const [account, setAccount] = useState(null);
const [isConnected, setIsConnected] = useState(false);
const [votingStatus, setVotingStatus] = useState(true);
const [remainingTime, setremainingTime] = useState("");
const [candidates, setCandidates] = useState([]);
const [number, setNumber] = useState("");
const [CanVote, setCanVote] = useState(true);
const [winnerName, setWinnerName] = useState("");
const [winnerVoteCount, setWinnerVoteCount] = useState(0);

useEffect(() => {
getCandidates();
getRemainingTime();
getCurrentStatus();
if (window.ethereum) {
window.ethereum.on("accountsChanged", handleAccountsChanged);
}

return () => {
if (window.ethereum) {
window.ethereum.removeListener(
"accountsChanged",
handleAccountsChanged
);
}
};
});

async function vote() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);

const tx = await contractInstance.vote(number);
await tx.wait();
canVote();
}
```

```
async function canVote() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);
const voteStatus = await contractInstance.voters(await signer.getAddress());
setCanVote(voteStatus);
}

async function getCandidates() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);
const candidatesList = await contractInstance.getAllVotesOfCandiates();
const formattedCandidates = candidatesList.map((candidate, index) => {
return {
index: index,
name: candidate.name,
voteCount: candidate.voteCount.toNumber(),
};
});
setCandidates(formattedCandidates);
}

async function getCurrentStatus() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);
const status = await contractInstance.getVotingStatus()
```

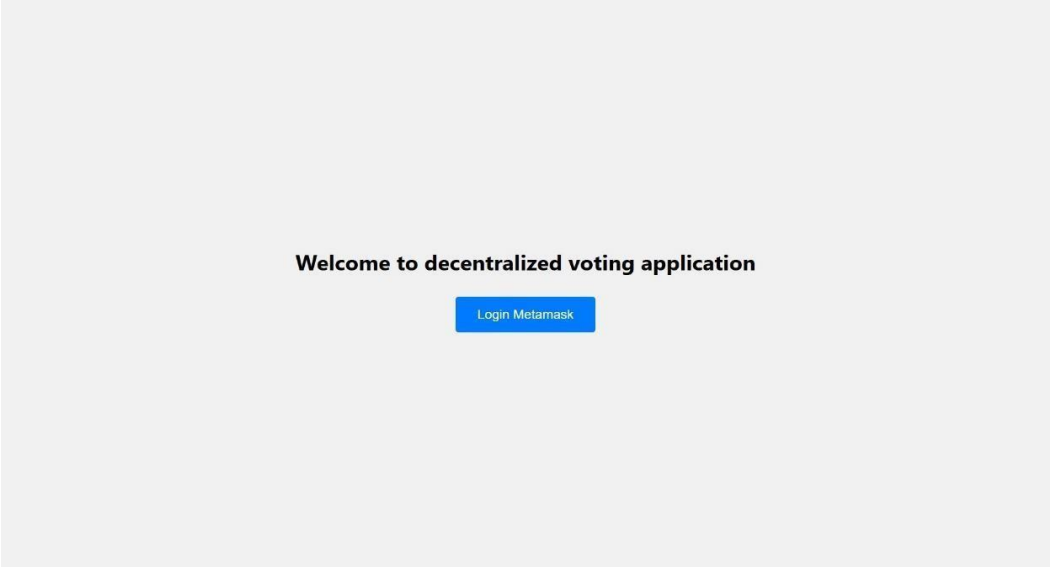
```
setVotingStatus(status);
}

async function getRemainingTime() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);
const time = await contractInstance.getRemainingTime();
setremainingTime(time.toNumber());
}

function handleAccountsChanged(accounts) {
if (accounts.length > 0 && account !== accounts[0]) {
setAccount(accounts[0]);
canVote();
} else {
setIsConnected(false);
setAccount(null);
}
}

async function connectToMetamask() {
if (window.ethereum) {
try {
const provider = new ethers.providers.Web3Provider(window.ethereum);
setProvider(provider);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const address = await signer.getAddress();
setAccount(address);
console.log("Metamask Connected : " + address);
setIsConnected(true);
canVote();
} catch (err) {
console.error(err);
}
} else {
alert("Metamask is not detected in the browser");}
```

```
}

async function handleNumberChange(e) {
setNumber(e.target.value);
}

async function getWinner() {
const provider = new ethers.providers.Web3Provider(window.ethereum);
await provider.send("eth_requestAccounts", []);
const signer = provider.getSigner();
const contractInstance = new ethers.Contract(
contractAddress,
contractAbi,
signer
);

// Retrieve winner details
try {
const [name, voteCount] = await contractInstance.getWinner();
setWinnerName(name);
setWinnerVoteCount(voteCount.toNumber());
} catch (error) {
console.error("Error fetching winner:", error);
}
}

useEffect(() => {
if (!votingStatus) {
getWinner();
}
}, [votingStatus]);
return (
<div className="App">
{votingStatus ? (
isConnected ? (
<Connected
account={account}
candidates={candidates}
remainingTime={remainingTime}
number={number}
handleNumberChange={handleNumberChange}
voteFunction={vote}
showButton={CanVote}
```

```
/>
) : (
<Login connectWallet={connectToMetamask} />
)
) : (
<Finished winnerName={winnerName} winnerVoteCount={winnerVoteCount} />
)}
</div>
);
}
export default App;
```

OUTPUT

```
>npm start

Compiled successfully!

You can now view asset in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.211.2:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully .
```



You are Connected to Metamask

Metamask Account: 0x8b0A7f802FEbf95eDC1A64De858f6774A1dB3055

Remaining Time: 1284

Entern Candidate Index

Vote

Index	Candidate name	Candidate votes
0	AAA	0
1	BBB	0
2	CCC	0
3	DDD	0

You are Connected to Metamask

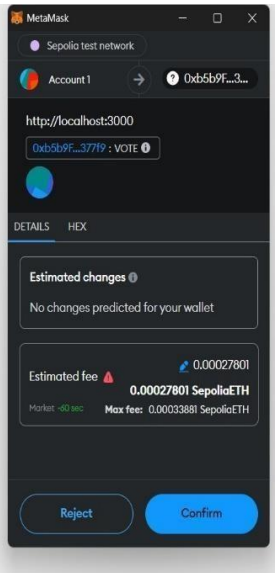
Metamask Account: 0x8b0A7f802FEbf95eDC1A64De858f6774A1dB3055

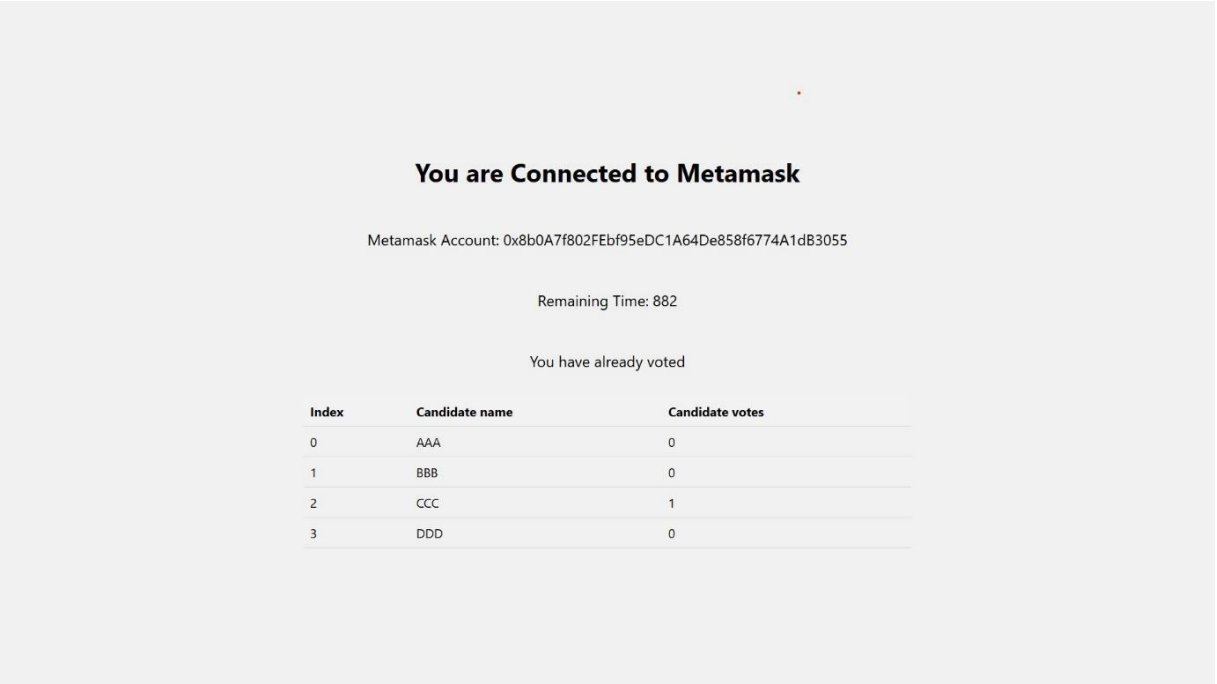
Remaining Time: 1074

2

Vote

Index	Candidate name	Candidate votes
0	AAA	0
1	BBB	0
2	CCC	0
3	DDD	0





RESULT

Thus, the blockchain-based voting application is successfully created and deployed, ensuring a secure and transparent voting process while providing a platform to test blockchain transactions and rules**.**

EX. NO: 04	DEPLOY AN ASSET-TRANSFER APP USING BLOCKCHAIN.
DATE:	

AIM

The aim of this experiment is to create and deploy an Asset-Transfer App using the Ethereum Blockchain Network, specifically on the Volta test network, using Hardhat. This app will enable the secure and transparent transfer of assets between parties.

PROCEDURE

1. **Prerequisites**

Before starting, ensure you have the following installed:

1. Node.js and npm
2. Hardhat
3. MetaMask (browser extension for managing Ethereum wallets)
4. Git

1. **Setting Up the Development Environment**

a. Clone the Repository:

```
> git clone https://github.com/IT-Blockchain-Exp/asset-Transfer-App.git
  // (package details)
> cd asset-Transfer-App
```

b. Install Dependencies and Set Up Hardhat:

```
>npm i
> npx hardhat
```

2. **Smart Contract Development**

a. Create the Smart Contract

Navigate to the contracts directory and create a new file named TransferSepoliaEth.sol and Write the smart contract code to handle asset transfers.

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

contract TransferSepoliaETH {

    event Transfer(address indexed from, address indexed to, uint256 amount);

    function transferEth(address payable _to, uint256 _amount) public payable
{
        require(msg.value == _amount, "Must send the exact amount of ETH");
        // Transfer the amount to the specified address
        _to.transfer(_amount);
        // Emit the transfer event
        emit Transfer(msg.sender, _to, _amount);
    }
}
```

b. Compile the smart contract

```
>npx hardhat compile
```

3. Deploying the Smart Contract

a. Create a Deployment Script

Navigate to the scripts directory and create a new file named deploy.js.

```
async function main() {
    const AssetTransfer = await ethers.getContractFactory("AssetTransfer");
    const assetTransfer = await AssetTransfer.deploy();
    await assetTransfer.deployed();
    console.log("AssetTransfer deployed to:", assetTransfer.address);
}

main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
        process.exit(1);
    });
```

b. Deploy the Contract to the Volta Network

Modify the hardhat.config.js to include configurations for the Volta network.

```
require("@nomiclabs/hardhat-waffle");
require("@nomiclabs/hardhat-ethers");

const API_URL = "https://volta-rpc.energyweb.org/";
const PRIVATE_KEY = ''YOUR-PRIVATE-KEY'';

module.exports = {
 solidity: "0.8.0", // Specify the Solidity version
 networks: {
   volta: {
     url: API_URL || "https://volta-rpc.energyweb.org/",
     accounts: [`0x${PRIVATE_KEY}`]
   }
 }
};
```

c. Deploy the smart contract to the Volta network

```
>npx hardhat run scripts/deploy.js --network volta
```

5. Setting Up the Blockchain Network

a. Connect MetaMask to Volta Network

Open MetaMask and add a new network with the following settings:

i. Network Name: Volta
ii. New RPC URL: https://volta-rpc.energyweb.org
iii. Chain ID: 73799
iv. Currency Symbol: volta

b. Import Accounts to MetaMask

Import the accounts used in your Hardhat configuration by using the private keys.

6. Developing the Frontend

a. Setting Up React App

Navigate to the client directory and install necessary dependencies and start the React development server.

```
>cd src
>npm install
>npm start
```

7. Connecting to the Smart Contract

Use the ethers.js library to connect to the Ethereum network and interact with the deployed smart contract, also update the App.js file to include logic for transferring assets and minting new assets.

```
import React, { useState } from 'react';
import { ethers } from 'ethers';
import { Link } from 'react-router-dom';
import { LuSend } from "react-icons/lu";

const contractAddress = "<CONTRACT_ADDRESS>";
const contractABI = [
"function transferEth(address payable _to, uint256 _amount) public payable"
];

const Wallet = ({ account, setAccount }) => {
const [recipientAddress, setRecipientAddress] = useState('');
const [ethAmount, setEthAmount] = useState('');
const [isLoading, setIsLoading] = useState(false);
const [showTransferForm, setShowTransferForm] = useState(false);

const connectWallet = async () => {
if (window.ethereum) {
try {
const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
setAccount(accounts[0]);
console.log("Connected account:", accounts[0]);
} catch (error) {
console.error("Connection error:", error);
}
} else {
```

```
alert("MetaMask not detected");
}
};

const isValidAddress = (address) => {
return ethers.isAddress(address);
};

const transferVoltaETH = async () => {
if (!recipientAddress || !ethAmount) {
alert("Please enter both recipient address and amount.");
return;
}

if (!isValidAddress(recipientAddress)) {
alert("Invalid recipient address.");
return;
}

if (parseFloat(ethAmount) <= 0) {
alert("Amount must be greater than zero.");
return;
}

setIsLoading(true);
try {
const amountInWei = ethers.parseUnits(ethAmount, 18);
const provider = new ethers.BrowserProvider(window.ethereum);
const signer = await provider.getSigner();
const contract = new ethers.Contract(contractAddress, contractABI, signer);

const transaction = await contract.transferEth(recipientAddress, amountInWei, {
value: amountInWei,
gasLimit: 500000,
});

await transaction.wait();
alert(`Successfully sent ${ethAmount} ETH to ${recipientAddress}`);


const transferDetails = {
from: account,
to: recipientAddress,
amount: ethAmount,
timestamp: new Date().toLocaleString(),
```

```
};
const storedTransfers = JSON.parse(localStorage.getItem('transfers')) || [];
storedTransfers.push(transferDetails);
localStorage.setItem('transfers', JSON.stringify(storedTransfers)); // Update local storage

// Reset fields after transaction
setRecipientAddress('');
setEthAmount('');
} catch (error) {
console.error("Transaction failed:", error);
alert("Transaction failed, see console for details.");
} finally {
setIsLoading(false);
}
};

return (
<div className="flex justify-center items-center min-h-screen  bg-customdark text-white">
<div className="text-gray-800 shadow-md rounded-lg p-8 max-w-lg w-full bg-white bg-opacity-20">
<h2 className="text-3xl font-bold text-center mb-6 text-white">Asset Transfer</h2>

{!account ? (
<button className="w-full border-2 border-gray-300 text-white py-3 rounded-lg hover:text-black font-bold hover:border-black" onClick={connectWallet}>
Connect MetaMask
</button>
) : (
<div>
{!showTransferForm ? (
<div className="flex space-x-4 mb-4">
<button
className="w-full bg-green-500 text-white py-3 rounded-lg hover:bg-green-600 font-bold flex items-center justify-center"
onClick={() => setShowTransferForm(true)}
>
<LuSend className="text-2xl mr-2" /> {/* This will make the icon larger and add margin to the right */}
<div>Send</div>
</button>
<Link to="/transfer-history" className="w-full border-2 border-gray-300 text-white py-3 rounded-lg hover:text-black font-bold hover:border-black text-center">
View Asset Transfers
</Link>
```

```
</div>
) : (
<div className="mb-6">
<h3 className="text-lg font-semibold mb-4 text-white">Transfer VoltaETH</h3>
<p className="mb-4 text-white">Connected Account: <span className="font-
bold">{account}</span></p>
<input
type="text"
placeholder="Recipient Address"
className="w-full border bg-transparent border-gray-300 rounded-lg p-3 mb-4 text-white"
value={recipientAddress}
onChange={(e) => setRecipientAddress(e.target.value)}
/>
<input
type="text"
placeholder="Amount in Volta"
className="w-full border bg-transparent border-gray-300 rounded-lg p-3 mb-6 text-white"
value={ethAmount}
onChange={(e) => setEthAmount(e.target.value)}
/>
<div className="flex flex-col items-center space-y-4">
<button className="w-[250px] bg-green-500 text-white py-3 rounded-lg hover:bg-green-
600" onClick={transferVoltaETH} disabled={isLoading}>
{isLoading ? 'Sending...' : 'Send Volta'}
</button>
<button
className="w-[250px] bg-gray-500 text-white py-3 rounded-lg hover:bg-gray-600"
onClick={() => setShowTransferForm(false)}
>
Cancel
</button>
</div>
</div>
)}
</div>
)}
</div>
</div>
);
};

export default Wallet;
```

OUTPUT

>npm start

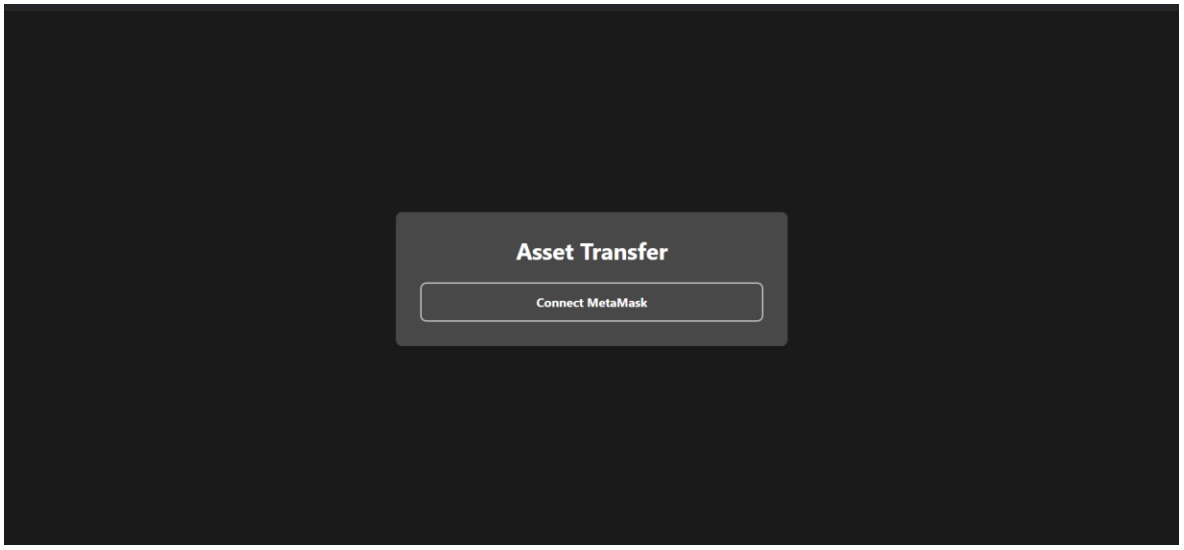
Compiled successfully!

You can now view asset in the browser.

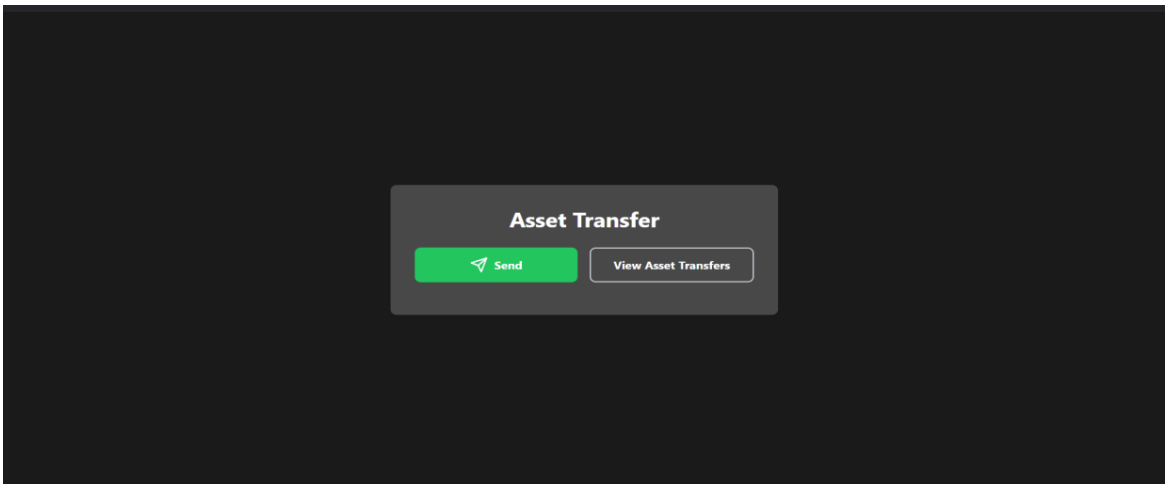
Local: http://localhost:3000
On Your Network: http://192.168.211.2:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

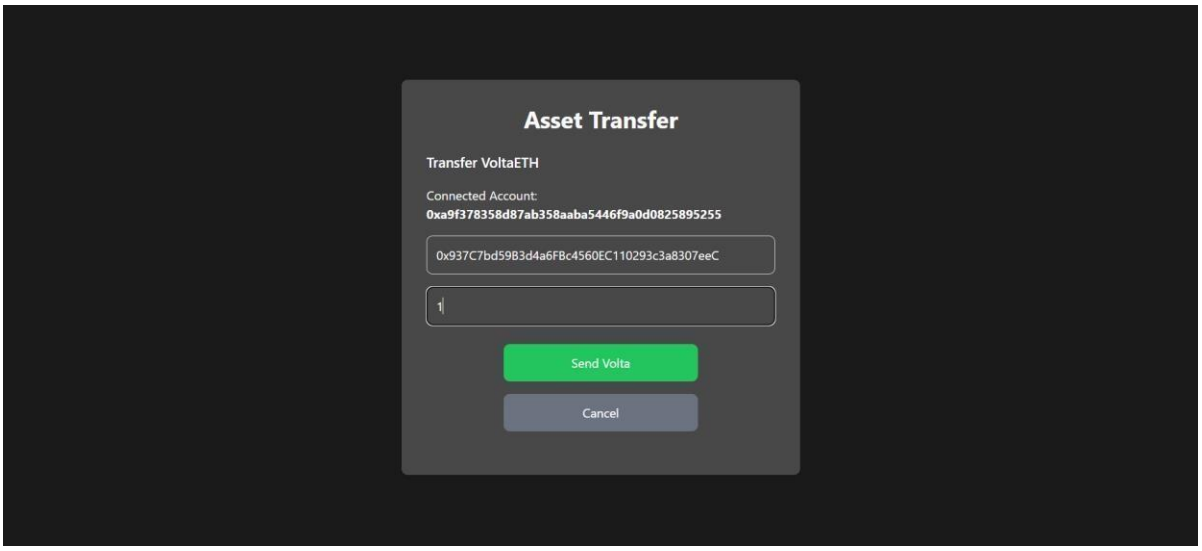
webpack compiled successfully



After compiling your application, a browser window will pop up. You will need to approve the connection with MetaMask to enable interaction with the Asset Transfer App.



- **Send**: This button initiates the transaction process to transfer assets to a specified recipient.
- **History**: Clicking this button allows you to view the transaction history of previous asset transfers.



- **Specify Recipient Wallet Address**: Input the wallet address of the merchant account you wish to send assets to.
- **Amount**: Indicate the amount of Volta coins you want to transfer.

By accessing the "View Asset Transfer" page, you can display the transaction history. This feature allows users to track all asset transfers made within the application, providing transparency and accountability.

RESULT

Thus, the asset-transfer app using blockchain is created and deployed successfully using the Ethereum network by transferring the volta coin.

EX. NO: 05	FITNESS CLUB REWARDS USING BLOCKCHAIN TECHNOLOGY
DATE:	

AIM

To create a blockchain-based rewards system for a fitness club, allowing secure member management and reward allocation using MetaMask.

PROCEDURE

1. **Prerequisites**

Before starting , ensure you have the following installed:

1. Metamask (Browser extension)
2. Live server (Extension on VsCode)

2. **Setting up the environment**

a. Clone the Repository:

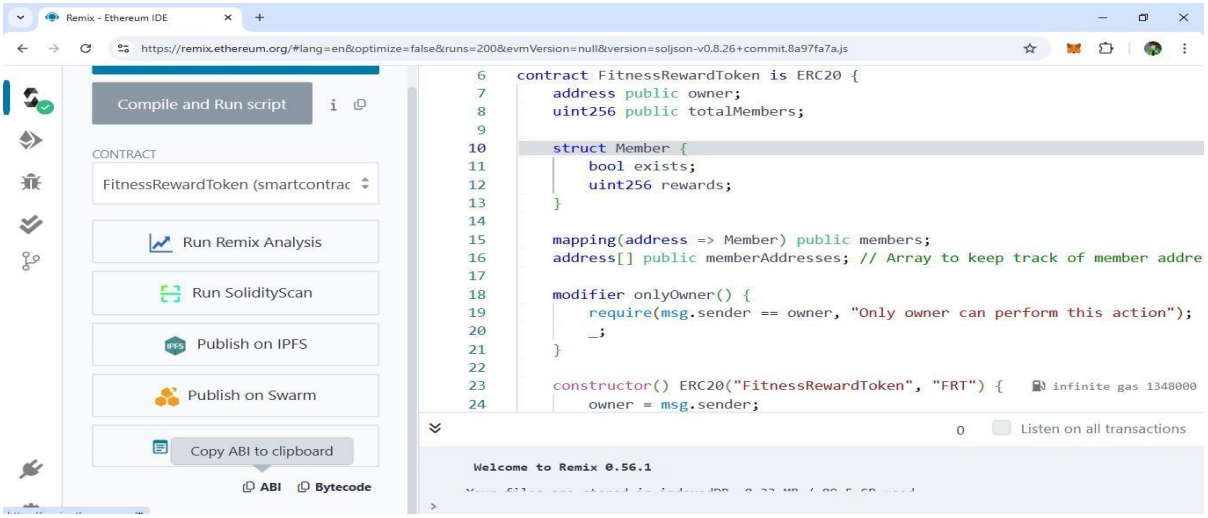
(package details are available in repository)

>git clone https://github.com/IT-Blockchain-Exp/fitness-Reward-App.git

>cd fitness-Reward-App

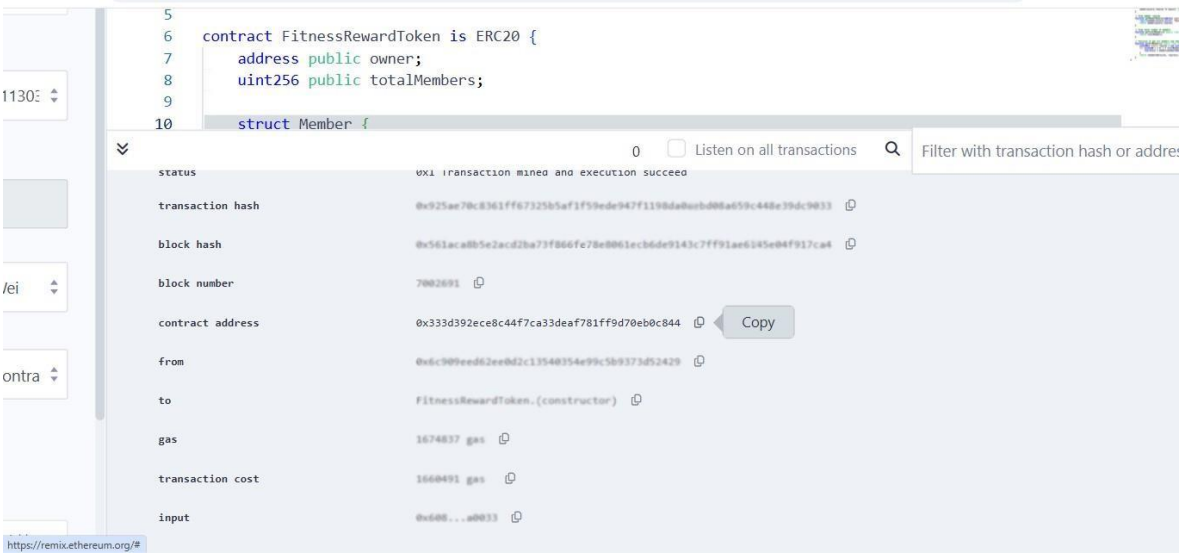
b. Compile the smart contract:

- Open remix ide in web browser.
- Upload the file in the editor.
- Compile the file.
- Copy the ABI provided after compilation.
- Replace the ABI in the contractABI in index.html file.



c. Deploy the smart contract:

- Select deploy and run transactions from the side bar.
- Choose **Injected Provider – Metamask** in environment.
- Deploy the smart contract.
- Copy the contract address from the console.
- Place it in the contractAddress in index.html file.



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

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract FitnessRewardToken is ERC20 {
address public owner;

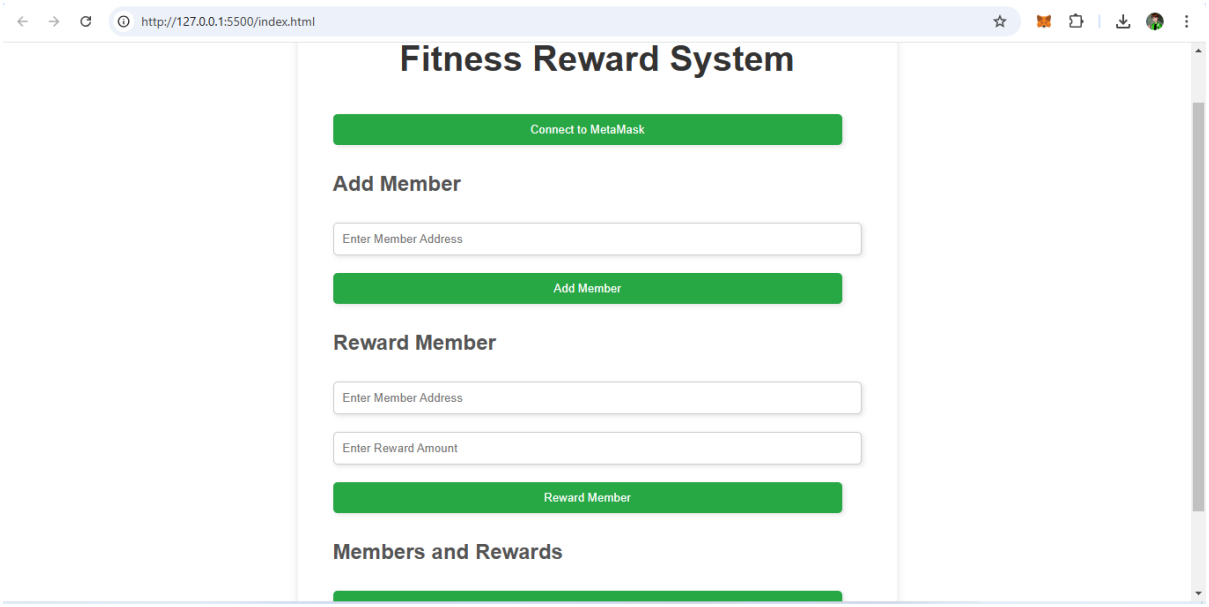
```
    uint256 public totalMembers;

    struct Member {
        bool exists;
        uint256 rewards;
    }

    mapping(address => Member) public members;
    address[] public memberAddresses; // Array to keep track of member addresses

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can perform this action");
        _;
    }

    constructor() ERC20("FitnessRewardToken", "FRT") {
        owner = msg.sender;
        _mint(owner, 1000000 * 10 ** decimals()); // Initial supply to owner
        totalMembers = 0;
    }

    // Add new member
    function addMember(address user) public onlyOwner {
        require(!members[user].exists, "User is already a member");
        members[user] = Member(true, 0); // New member with 0 initial rewards
        memberAddresses.push(user); // Add the new member's address to the list
        totalMembers++;
    }

    // Reward member
    function rewardUser(address user, uint256 amount) public onlyOwner {
        require(members[user].exists, "User is not a member");
        _transfer(owner, user, amount); // Transfer tokens from owner to user
        members[user].rewards += amount; // Update user's reward balance
    }

    // View member rewards
    function getMemberRewards(address user) public view returns (uint256) {
        require(members[user].exists, "User is not a member");
        return members[user].rewards;
    }

    // View total number of members
    function getTotalMembers() public view returns (uint256) {
        return totalMembers;
    }

    // Function to get all members and their rewards
    function getAllMembers() public view returns (address[] memory, uint256[]
memory) {
        uint256[] memory rewards = new uint256[](totalMembers);
```

```
        for (uint256 i = 0; i < totalMembers; i++) {
            rewards[i] = members[memberAddresses[i]].rewards;
        }
        return (memberAddresses, rewards);
    }
}
```

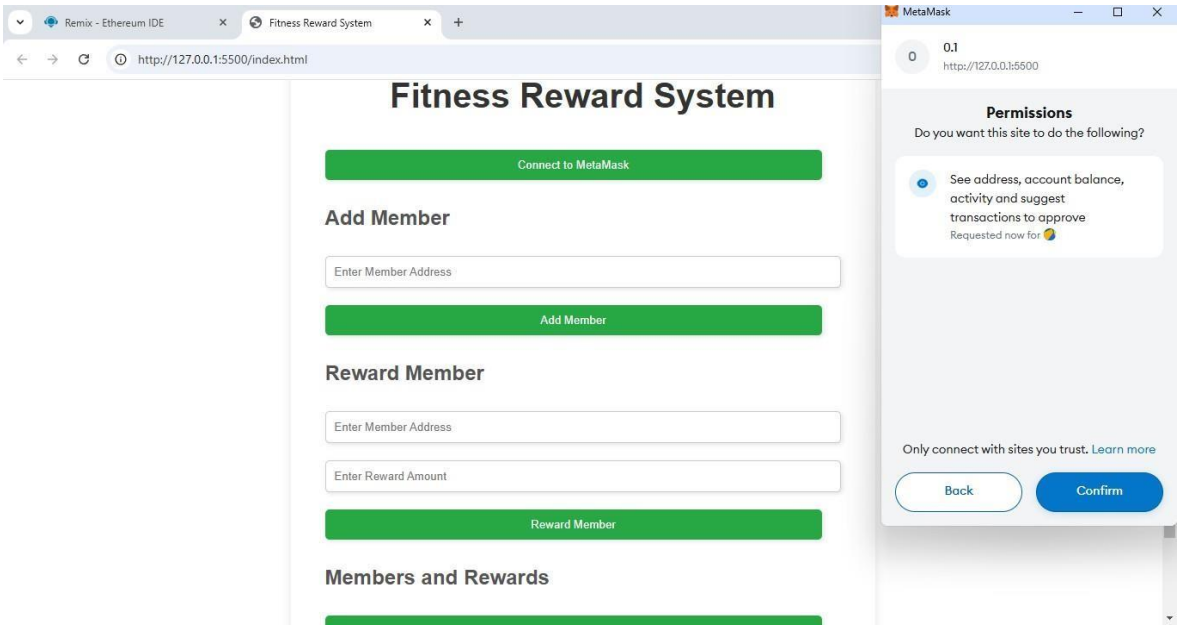
d. Executing the dapp:

- Open HTML file with Live Server.

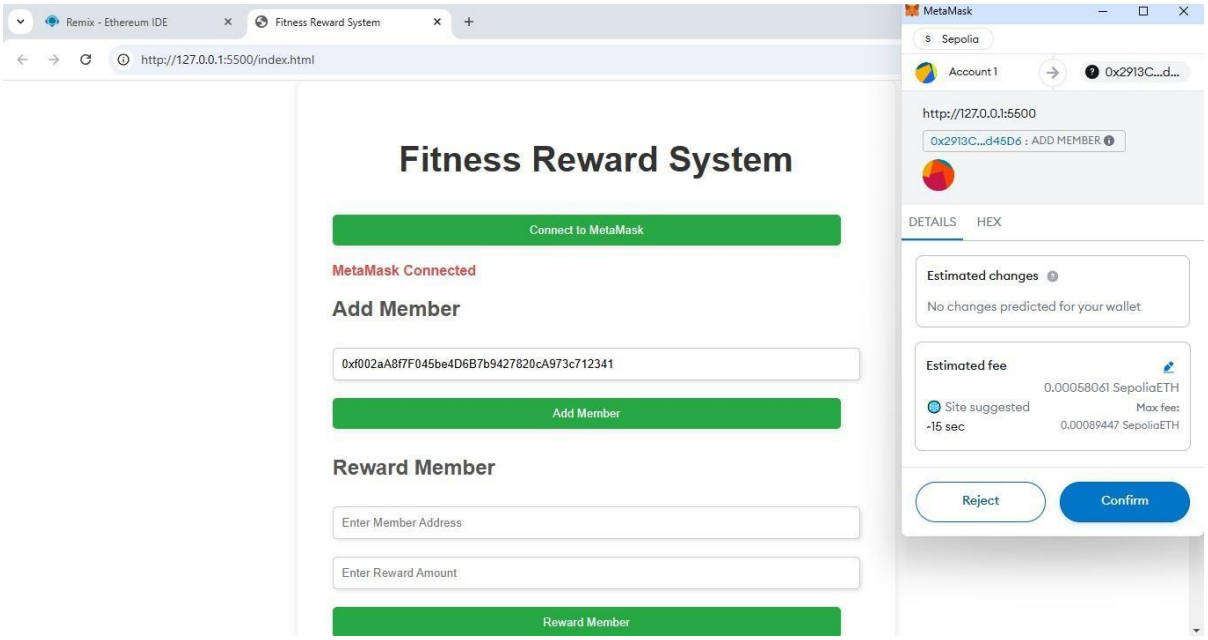
OUTPUT

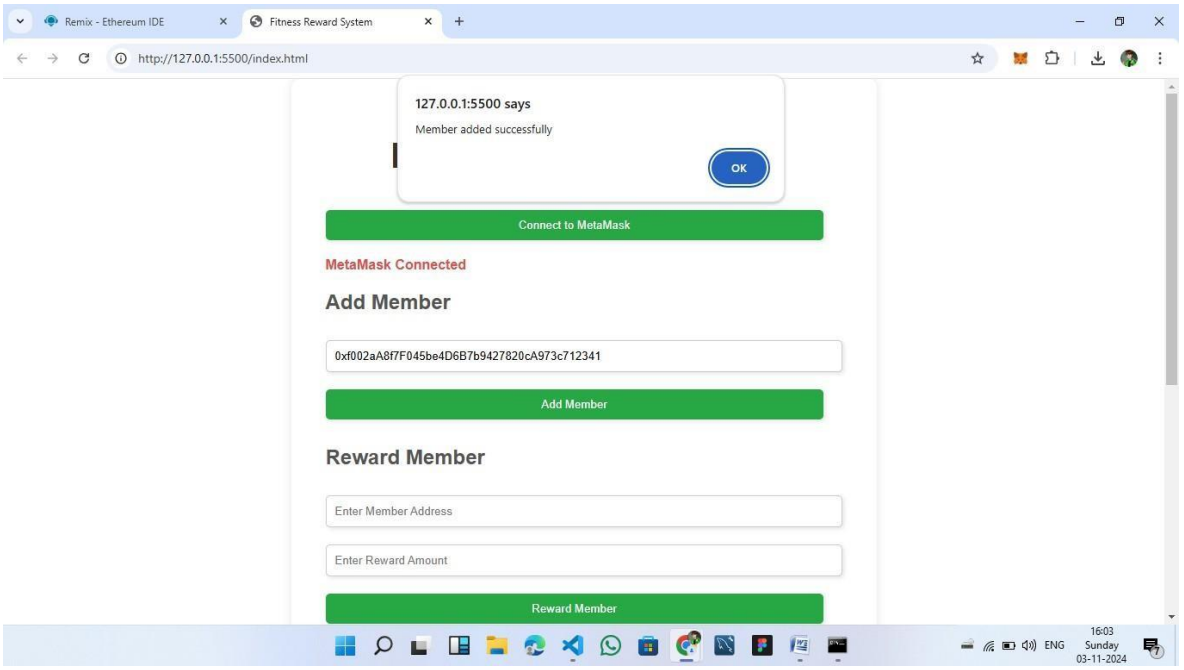


On opening with live server , a browser window will pop up. For proceeding with transactions, first connect to metamask.

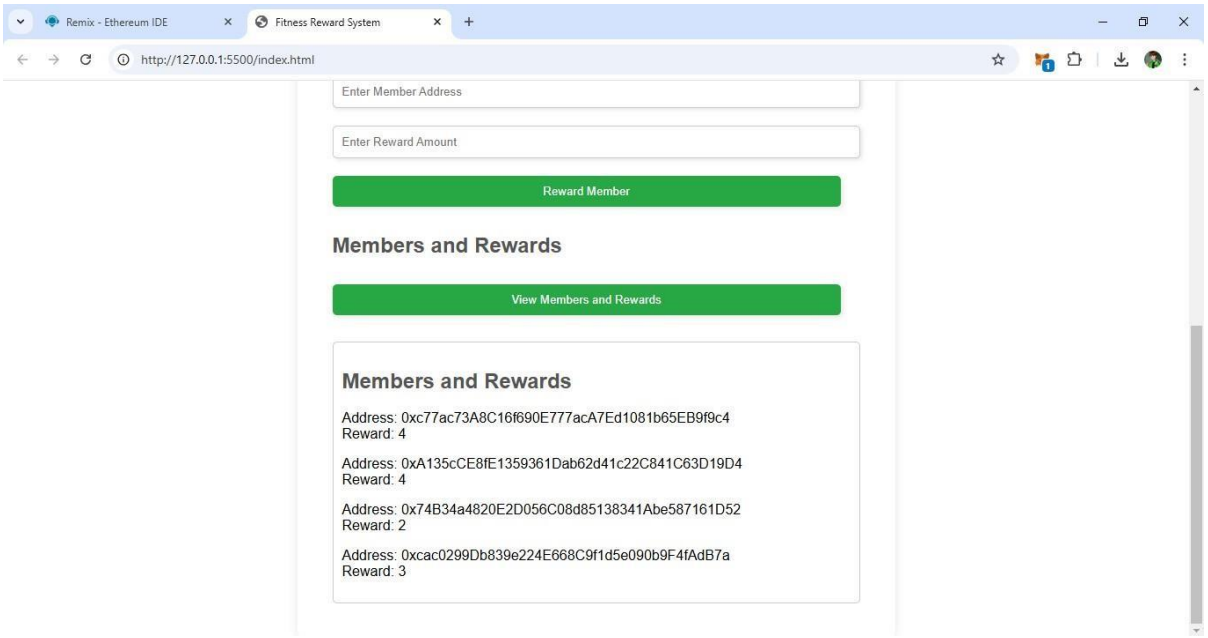


➢ Add member

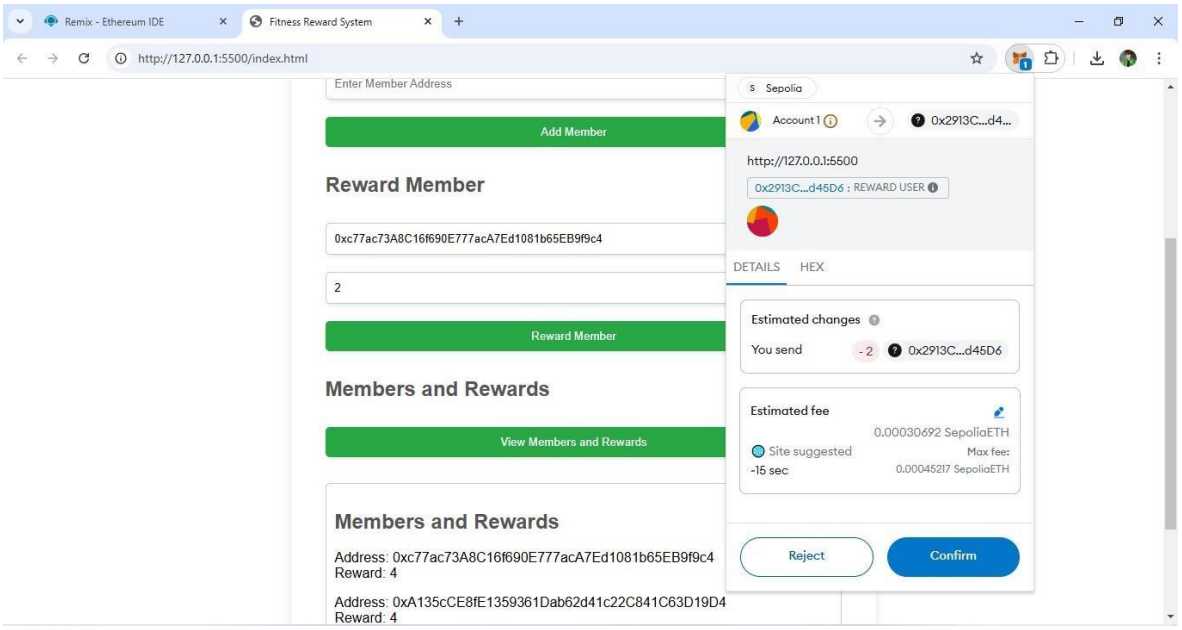

Remix - Ethereum IDE
Fitness Reward System
http://127.0.0.1:5500/index.html
Fitness Reward System
Connect to MetaMask
MetaMask Connected
Add Member
0xf002aA8f7F045be4D6B7b9427820cA973c712341
Add Member
Reward Member
Enter Member Address
Enter Reward Amount
Reward Member
MetaMask
Sepolia
Account 1
0x2913C...d...
http://127.0.0.1:5500
0x2913C...d45D6 : ADD MEMBER
DETAILS
HEX
Estimated changes
No changes predicted for your wallet
Estimated fee
0.00058061 SepoliaETH
Site suggested
Max fee:
-15 sec
0.00089447 SepoliaETH
Reject
Confirm

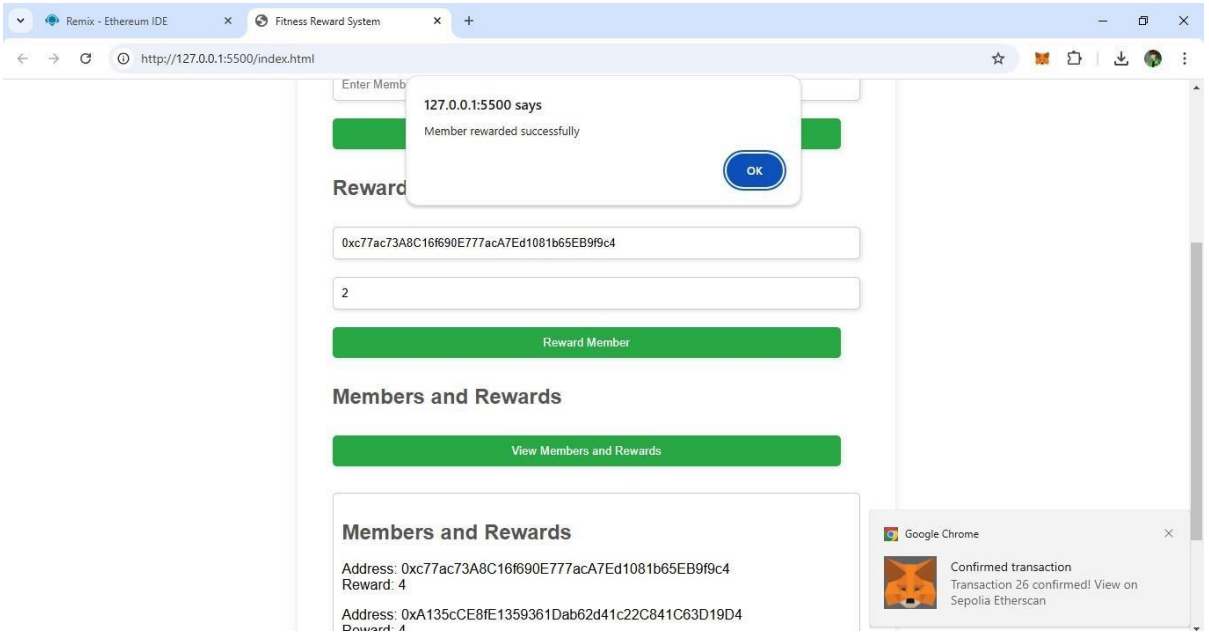

Remix - Ethereum IDE
Fitness Reward System
http://127.0.0.1:5500/index.html
127.0.0.1:5500 says
Member added successfully
OK
Connect to MetaMask
MetaMask Connected
Add Member
0xf002aA8f7F045be4D6B7b9427820cA973c712341
Add Member
Reward Member
Enter Member Address
Enter Reward Amount
Reward Member
ENG
16:03
Sunday
03-11-2024

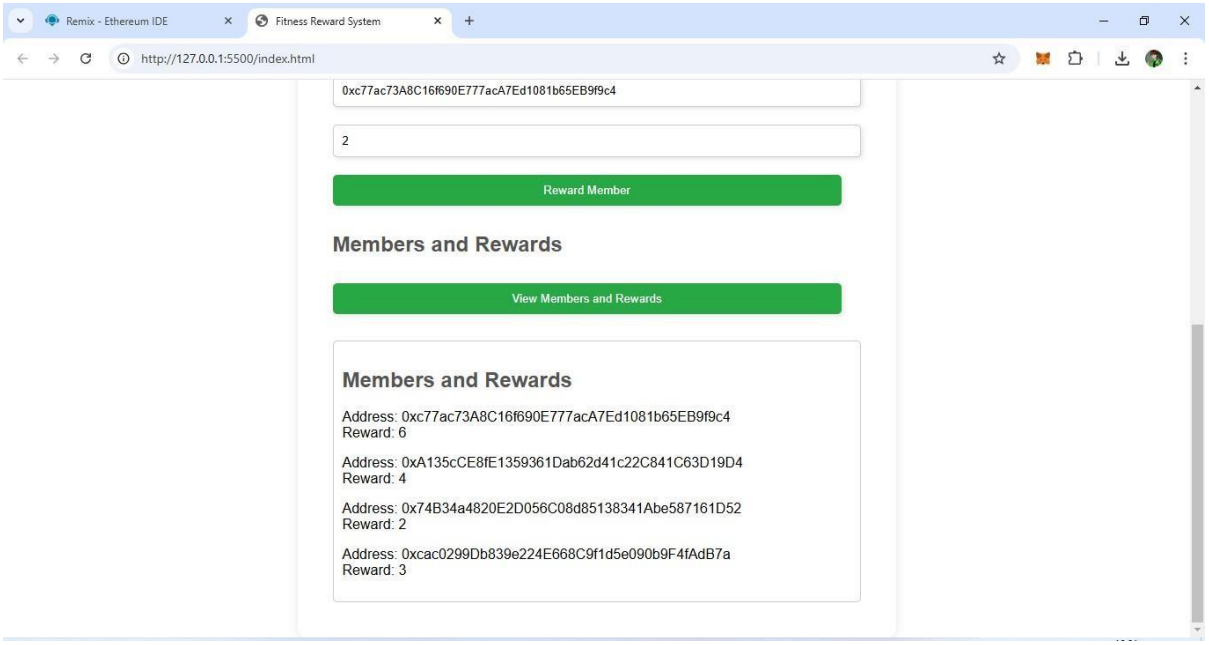
- **View members and their rewards**



- **Reward members**







RESULT

Thus, a fitness reward system has been developed using blockchain technology.

EX. NO: 06	CAR AUCTION NETWORK
DATE:	

AIM

The aim of this experiment is to create and deploy a secure Car Auction network on the Ethereum Blockchain Network that ensures the transparency and immutability of Auction and bidding processes, while enabling interaction with the blockchain to execute transactions and test network rules.

PROCEDURE

1. Prerequisites

Before starting, ensure you have the following installed:

1. VS Code (With **live server** extension)
2. MetaMask
3. Git

2. Setting Up the Project Folder

a. Clone the Repository:

> git clone https://github.com/IT-Blockchain-Exp/car-Auction-App.git (package details)

> cd car-Auction-App

3. Smart Contract Development

a. Create the Smart Contract

Navigate to the contracts directory and create a new file named auction.sol. Write the smart contract code to handle voting-process.

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

import "@openzeppelin/contracts/access/Ownable.sol";
```

```
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

contract CarAuctionNetwork is ReentrancyGuard {
    using Counters for Counters.Counter;

    struct Car {
        string make;
        string model;
        uint256 year;
        address payable owner;
        bool registered;
    }

    struct Auction {
        uint256 carId;
        address payable seller;
        uint256 minBid;
        uint256 highestBid;
        address payable highestBidder;
        bool ended;
        bool started;
    }

    Counters.Counter private _carIdCounter;
    mapping(uint256 => Car) public cars;
    mapping(uint256 => Auction) public carAuctions;

    event CarRegistered(uint256 carId, string make, string model, uint256 year,
address owner);
    event AuctionStarted(uint256 carId, uint256 minBid);
    event NewBid(uint256 carId, address bidder, uint256 bidAmount);
    event AuctionEnded(uint256 carId, address winner, uint256 bidAmount);

    // Register a car to the auction system
    function registerCar(string memory make, string memory model, uint256 year)
public returns (uint256) {
        _carIdCounter.increment();
        uint256 carId = _carIdCounter.current();

        cars[carId] = Car({
```

```
            make: make,
            model: model,
            year: year,
            owner: payable(msg.sender),
            registered: true
        });

        emit CarRegistered(carId, make, model, year, msg.sender);
        return carId;
    }

    // Start an auction for a registered car
    function startCarAuction(uint256 carId, uint256 minBid) public {
        Car storage car = cars[carId];
        require(car.registered, "Car not registered");
        require(car.owner == msg.sender, "You do not own this car");
        require(!carAuctions[carId].started, "Auction already started for this car");

        carAuctions[carId] = Auction({
            carId: carId,
            seller: car.owner,
            minBid:  minBid,
            highestBid: 0,
            highestBidder: payable(address(0)),
            ended: false,
            started: true
        });

        emit AuctionStarted(carId, minBid);
    }

    // Place a bid for a car auction
    function placeBid(uint256 carId) public payable nonReentrant {
        Auction storage auction = carAuctions[carId];
        require(auction.started, "Auction has not started");
        require(!auction.ended, "Auction has already ended");
        require(msg.value >= auction.minBid, "Bid does not meet minimum bid");
        require(msg.value > auction.highestBid, "Bid must be higher than the current
highest bid");

        // Refund the previous highest bidder
```

```
        if (auction.highestBidder != address(0)) {
            auction.highestBidder.transfer(auction.highestBid);
        }

        auction.highestBidder = payable(msg.sender);
        auction.highestBid = msg.value;

        emit NewBid(carId, msg.sender, msg.value);
    }

    // End the auction and transfer the car to the highest bidder
    function endAuction(uint256 carId) public nonReentrant {
        Auction storage auction = carAuctions[carId];
        require(auction.started, "Auction has not started");
        require(!auction.ended, "Auction has already ended");
        require(auction.seller == msg.sender, "Only the seller can end the auction");

        auction.ended = true;

        if (auction.highestBidder != address(0)) {
            // Transfer car ownership
            cars[carId].owner = auction.highestBidder;

            // Transfer funds to the seller
            auction.seller.transfer(auction.highestBid);
        }

        emit AuctionEnded(carId, auction.highestBidder, auction.highestBid);
    }

    // Get car details
    function getCarDetails(uint256 carId) public view returns (string memory make,
string memory model, uint256 year, address owner) {
        Car storage car = cars[carId];
        require(car.registered, "Car not registered");

        return (car.make, car.model, car.year, car.owner);
    }

    // Get auction details
```

```
    function getAuctionDetails(uint256 carId) public view returns (address seller,
uint256 minBid, uint256 highestBid, address highestBidder, bool ended) {
        Auction storage auction = carAuctions[carId];
        require(auction.started, "Auction has not started");

        return (auction.seller, auction.minBid, auction.highestBid, auction.highestBidder,
auction.ended);
    }

    // Get the total count of registered cars
    function getCarCount() public view returns (uint256) {
        return _carIdCounter.current(); // Return the current count of registered cars
    }

    // Fallback function to handle receiving Ether
    receive() external payable {}
}
```

b. Compile the smart contract

To compile the smart contract (auction.sol) that you created, go to https://remix.ethereum.org/, upload the contract, and then compile and deploy it in the IDE.

4. Deploying the Smart Contract

After the contract is deployed successfully, copy the contract addresss and assign it to the variable **contractAddress** in index.js

5. Setting Up the Blockchain Network

a. Create an account in MetaMask
b. Switch to Sepolia Testnet
c. Add some Sepolia ETH via free faucets available in internet (https://cloud.google.com/application/web3/faucet/ethereum/sepolia)

6. Developing the Frontend

a. Accessing HTML file

Select the **index.html** file and click on **Go live** in the VSCode and this would open the frontend in the browser.

7. Connecting to the Smart Contract

```
//Index.js
const contractAddress = "  "; //paste your contract address


// Ensure Web3 is initialized
if (typeof window.ethereum !== "undefined") {
  // Use window.ethereum as the provider
  web3 = new Web3(window.ethereum);

  // Request account access if needed
  try {
    window.ethereum.request({ method: "eth_requestAccounts" });
  } catch (error) {
    console.error("User denied account access", error);
  }
} else if (typeof window.web3 !== "undefined") {
  web3 = new Web3(window.web3.currentProvider);
  console.warn(
    "Using deprecated window.web3. Please upgrade to window.ethereum."
  );
} else {
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  console.warn("No Ethereum provider detected. Falling back to localhost.");
}

const contract = new web3.eth.Contract(contractABI, contractAddress);

// Register a new car
document.getElementById("registerCar").addEventListener("click", async () => {
const make = document.getElementById("make").value;
  const model = document.getElementById("model").value;
  const year = document.getElementById("year").value;

  if (make && model && year) {
    const accounts = await web3.eth.getAccounts();

    const estimatedGas = await contract.methods
      .registerCar(make, model, year)
      .estimateGas({ from: accounts[0] });
```

```
    const gasPrice = await web3.eth.getGasPrice();
    const reducedGasPrice = Math.floor(gasPrice * 0.9); // Ensure integer

    contract.methods
      .registerCar(make, model, year)
      .send({
        from: accounts[0],
        gas: estimatedGas,
        gasPrice: reducedGasPrice,
      })
      .then((receipt) => {
        alert("Car registered successfully!");
      })
      .catch((error) => {
        console.error(error);
      });
  } else {
    alert("Be sure to fill all the fields");
  }
});

// Start Auction
document.getElementById("startAuction").addEventListener("click", async () => {
const carId = document.getElementById("auctionCarId").value;
  const minBid = web3.utils.toWei(
    document.getElementById("minBid").value,
    "ether"
  );
  const accounts = await web3.eth.getAccounts();

  const estimatedGas = await contract.methods
    .startCarAuction(carId, minBid)
    .estimateGas({ from: accounts[0] });

  const gasPrice = await web3.eth.getGasPrice();
  const reducedGasPrice = Math.floor(gasPrice * 0.9);

  contract.methods
    .startCarAuction(carId, minBid)
    .send({
      from: accounts[0],
      gas: estimatedGas,
      gasPrice: reducedGasPrice,
```

```
    })
    .then((receipt) => {
      alert(`Auction started for car ID: ${carId}`);
    })
    .catch((error) => {
      console.error(error);
      alert(error);
    });
});

// Place a Bid
document.getElementById("placeBid").addEventListener("click", async () => {
const carId = document.getElementById("bidCarId").value;
  const bidAmount = web3.utils.toWei(
    document.getElementById("bidAmount").value,
    "ether"
  );
  const accounts = await web3.eth.getAccounts();

  const estimatedGas = await contract.methods
    .placeBid(carId)
    .estimateGas({ from: accounts[0], value: bidAmount });

  const gasPrice = await web3.eth.getGasPrice();
  const reducedGasPrice = Math.floor(gasPrice * 0.9);

  contract.methods
    .placeBid(carId)
    .send({
      from:  accounts[0],
      value:  bidAmount,
      gas: estimatedGas,
      gasPrice: reducedGasPrice,
    })
    .then((receipt) => {
      alert("Bid placed successfully!");
      console.log("then");
    })
    .catch((error) => {
      console.error(error);
      console.log("catch");
    });
});
```

```
// End Auction
document.getElementById("endAuction").addEventListener("click", async () => {
const carId = document.getElementById("endCarId").value;
  const accounts = await web3.eth.getAccounts();

  const estimatedGas = await contract.methods
    .endAuction(carId)
    .estimateGas({ from: accounts[0] });

  const gasPrice = await web3.eth.getGasPrice();
  const reducedGasPrice = Math.floor(gasPrice * 0.9);

  contract.methods
    .endAuction(carId)
    .send({
      from: accounts[0],
      gas: estimatedGas,
      gasPrice: reducedGasPrice,
    })
    .then((receipt) => {
      alert(`Auction ended for car ID: ${carId}`);
    })
    .catch((error) => {
      console.error(error);
    });
});

// Get Car and Auction Details of a single car
document.getElementById("getDetails").addEventListener("click", async () => {
const carId = document.getElementById("detailsCarId").value;

  if (carId.trim()) {
    // Hide the all-car table
    document.getElementById("allCarAuctionDetailsTable").style.display = "none";

    // Fetch Car Details
    let carDetails = await contract.methods
      .getCarDetails(carId)
      .call()
      .catch((error) => {
        console.error(error.message);
        alert(`Car ID: ${carId} is not registered`);
```

```
  });

// Fetch Auction Details
let auctionDetails = await contract.methods
  .getAuctionDetails(carId)
  .call()
  .catch((error) => {
    console.error(error.message);
    alert(`Auction not yet started for car ID: ${carId}`);
    carDetails = "";
  });

// If Car details exist, populate the table
if (carDetails) {
  const table = document.getElementById("carAuctionDetailsTable");
  const tbody = table.querySelector("tbody");

  // Clear existing rows
  tbody.innerHTML = "";

  // If auction details are not available, fill with "N/A"
  auctionDetails = auctionDetails || ["N/A", "N/A", "N/A", "N/A", false];

  // Add a new row with car and auction details
  const row = `
        <tr>
          <td>${carId}</td>
          <td>${carDetails[0]}</td>
          <td>${carDetails[1]}</td>
          <td>${carDetails[2]}</td>
          <td>${carDetails[3].slice(0, 5)}. .. ${carDetails[3].slice(
    carDetails[3].length - 4
  )}</td>
          <td>${auctionDetails[0].slice(
            0,
            5
          )}... ${auctionDetails[0].slice(
    auctionDetails[0].length - 4
  )}</td>
          <td>${
            auctionDetails[1] !== "N/A"
              ? web3.utils.fromWei(auctionDetails[1], "ether")
              : "N/A"
```

```
            }</td>
            <td>${
              auctionDetails[2] !== "N/A"
                ? web3.utils.fromWei(auctionDetails[2], "ether")
                : "N/A"
            }</td>
            <td>${auctionDetails[3].slice(
              0,
              5
            )}... ${auctionDetails[3].slice(
      auctionDetails[3].length - 4
    )}</td>
            <td>${auctionDetails[4] ? "Yes" : "No"}</td>
          </tr>
        `;

    tbody.insertAdjacentHTML("beforeend", row);
    table.style.display = "table"; // Show the table
   }
  } else {
   alert("Enter a valid car ID");
  }
});

// Get all cars and auction details
document.getElementById("getAllDetails").addEventListener("click", async () => {
  // Hide the single-car table when fetching all cars
  document.getElementById("carAuctionDetailsTable").style.display = "none";

  // Fetch total number of cars
  const carCount = await contract.methods.getCarCount().call();
  const table = document.getElementById("allCarAuctionDetailsTable");
  const tbody = table.querySelector("tbody");

  // Clear existing rows
  tbody.innerHTML = "";

  for (let i = 1; i <= carCount; i++) {
   try {
    // Fetch car details for each car
    const carDetails = await contract.methods.getCarDetails(i).call();

    // Default auction info
```

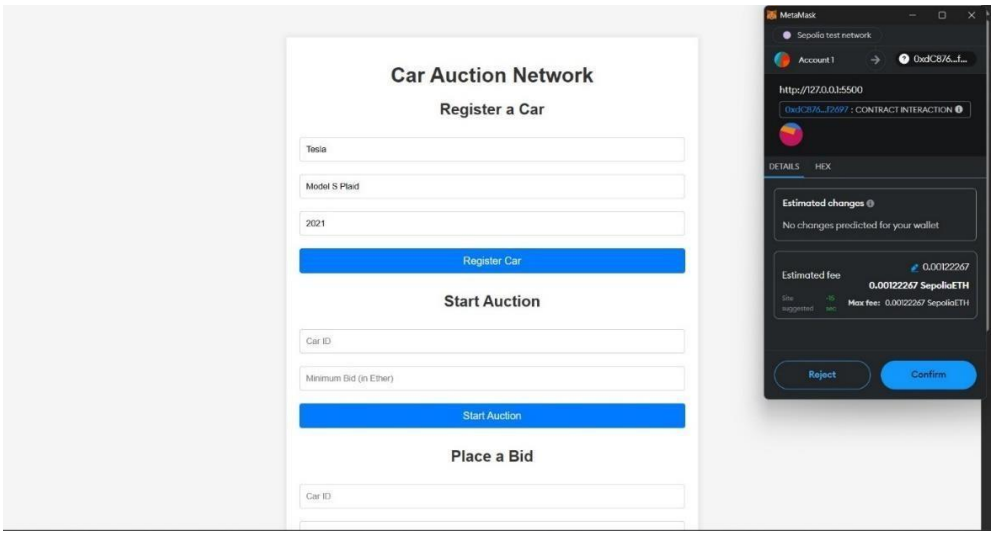
```
let auctionDetails;
let auctionStatus = "Auction Not Started";
let auctionInfo = {
 seller: "N/A",
 minBid: "N/A",
 highestBid: "N/A",
 highestBidder: "N/A",
 auctionEnded: false,
};

try {
 // Try fetching auction details (in case it exists)
 auctionDetails = await contract.methods.getAuctionDetails(i).call();

 // Check if the auction was started
 if (
  auctionDetails.seller !== "0x0000000000000000000000000000000000000000"
 ) {
  auctionInfo.seller = auctionDetails.seller;
  auctionInfo.minBid = web3.utils.fromWei(
  auctionDetails.minBid,
   "ether"
  );
  auctionInfo.highestBid =
   auctionDetails.highestBid > 0
    ? web3.utils.fromWei(auctionDetails.highestBid, "ether")
    : "No Bids yet";
  auctionInfo.highestBidder =
  auctionDetails.highestBidder !==
   "0x0000000000000000000000000000000000000000"
    ? auctionDetails.highestBidder
    : "0x0000000000000000000000000000000000000000";
  auctionInfo.auctionEnded = auctionDetails.ended;

  // Set auction status based on whether the auction has ended or not
  auctionStatus = auctionDetails.ended
   ? "Auction Completed"
   : "Auction Ongoing";
 }
} catch (auctionError) {
 console.log(`No auction started for car ID ${i}`);
}
```

```
// Add each car's details as a row
const row = `
  <tr>
    <td>${i}</td>
    <td>${carDetails[0]}</td>
    <td>${carDetails[1]}</td>
    <td>${carDetails[2]}</td>
    <td>${carDetails[3].slice(0, 5)}...${carDetails[3].slice(
  carDetails[3].length - 4
)}</td>
    <td>${
      auctionInfo.seller !== "N/A"
        ? auctionInfo.seller.slice(0, 5) +
          "..." +
          auctionInfo.seller.slice(auctionInfo.seller.length - 4)
        : "N/A"
    }</td>
    <td>${
      auctionInfo.minBid !== "N/A" ? auctionInfo.minBid + " ETH" : "N/A"
    }</td>
    <td>${
      auctionInfo.highestBid !== "N/A" &&
      auctionInfo.highestBid !== "No Bids yet"
        ? auctionInfo.highestBid + " ETH"
        : auctionInfo.highestBid
    }</td>
    <td>${
      auctionInfo.highestBidder !== "N/A"
        ? auctionInfo.highestBidder.slice(0, 5) +
          "..." +
          auctionInfo.highestBidder.slice(
            auctionInfo.highestBidder.length - 4
          )
        : "N/A"
    }</td>
    <td>${auctionStatus}</td>
  </tr>
`;

  tbody.insertAdjacentHTML("beforeend", row);
} catch (error) {
  console.error(`Error fetching details for car ID ${i}:`, error);
  const row = `
```

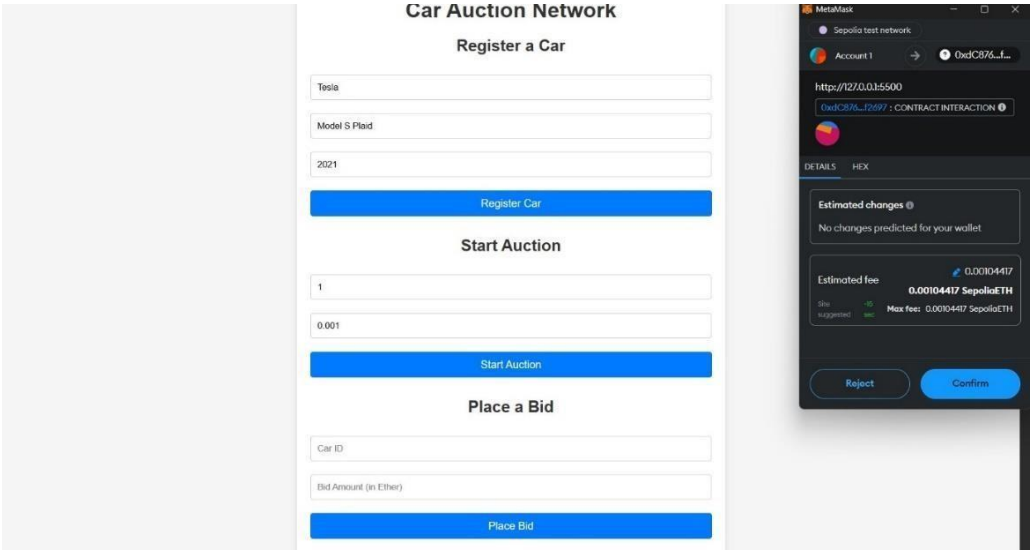
```
      <tr>
        <td>${i}</td>
        <td colspan="9">Error fetching details for this car.</td>
      </tr>
    `;
    tbody.insertAdjacentHTML("beforeend", row);
  }
 }

 table.style.display = "table"; // Show the table with all car details
});
```

OUTPUT

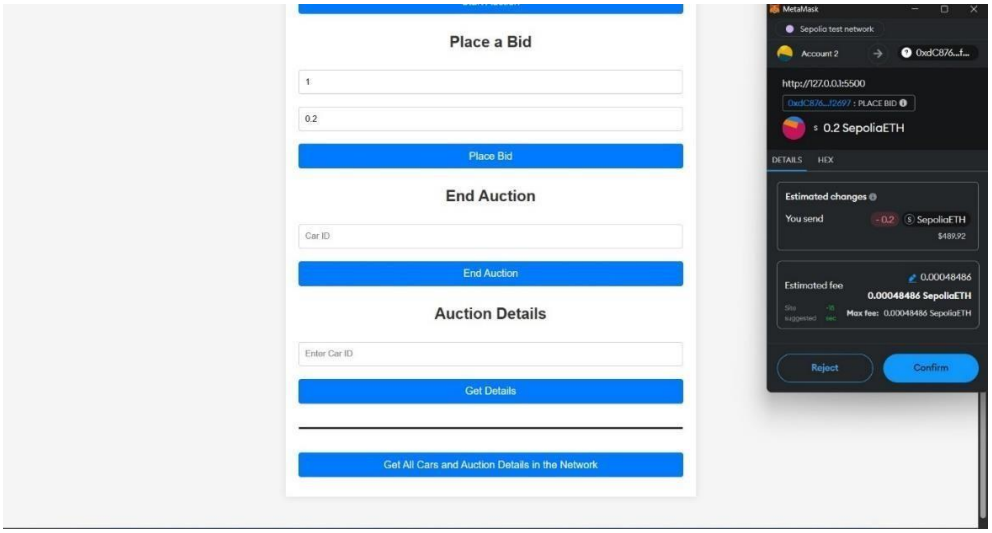
Register a Car



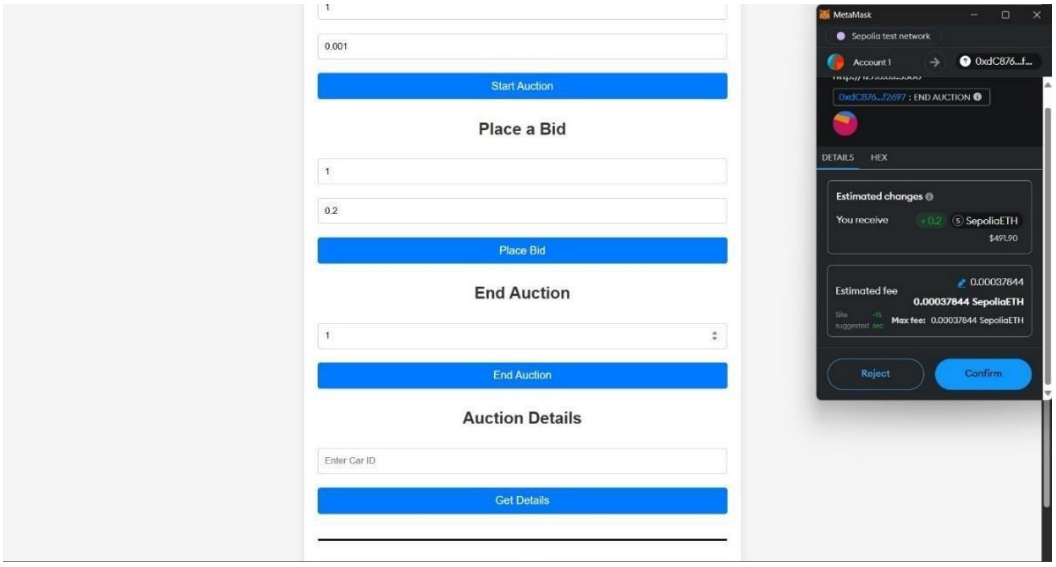
Start Auction



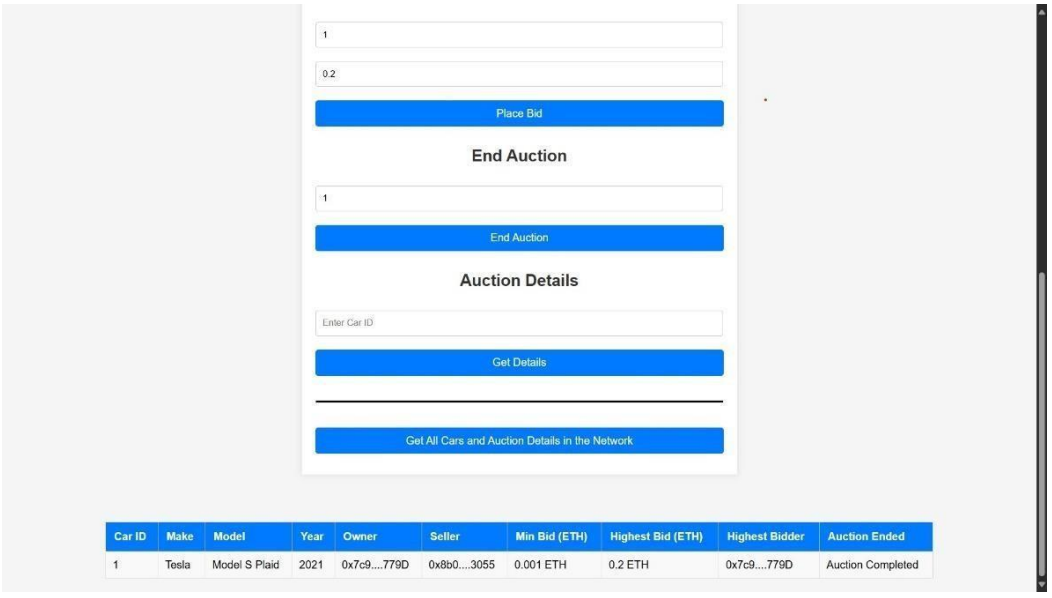
Place bid



End Auction



Retrieving details



RESULT

Thus, the blockchain-based car auction network is successfully created and deployed, ensuring a secure and transparent bidding process while providing a platform to test blockchain transactions and rules**.**