**Practical 7**

**Aim:** Build a decentralized Lottery application (DApp), Using ERC-20 Tokens that combine Ethereum's Web3 and Solidity smart contracts deploy on Test network.

**Background Information:**

**What's a Decentralized Lottery DApp?**

This DApp will manage a lottery system on the Ethereum blockchain, allowing participants to join using ERC-20 tokens.

1. **Lottery on Ethereum:**

Imagine a digital lottery where you buy tickets using special tokens that exist on Ethereum (a popular blockchain platform).

1. **Smart Contracts:**

* These are like digital rules that run automatically.
* In our lottery, smart contracts handle everything—ticket sales, picking winners, and giving out prizes.

**How it Works:**

1. **Creating Special Tokens (ERC-20):**

* These tokens are like virtual tickets.
* We create them using a special language called Solidity.
* These tokens follow specific rules that make them work smoothly on Ethereum.

1. **Building the Lottery's Brain (Smart Contracts):**

* With Solidity, we write code that forms the lottery.
* This code acts as the brain, keeping track of ticket sales, randomly picking winners, and making sure prizes go to the right people.

**Using** **Web3**:

1. **Linking to Ethereum:**

* Web3 is like a link that connects our lottery to the Ethereum network.
* It helps players interact with the lottery using their phones or computers.

1. **Buying Tickets & Checking Results:**

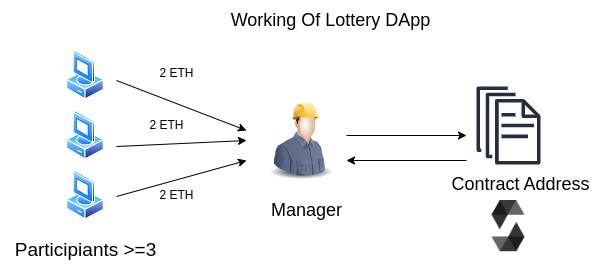
* Through Web3, players can buy tickets, see if they've won, and claim prizes—all while staying connected to the Ethereum network.

**Testing on a Playground (Test Network):**

1. **Practice Run**:

* Before launching on the real Ethereum network, we test the whole system on a network.
* It's like a safe playground where we try everything out without using real money.

1. **Ensuring It Works Smoothly:**



* This testing helps us catch any issues before the real launch, making sure the lottery runs smoothly and fairly for everyone.

Building a decentralized lottery app involves creating a digital lottery on Ethereum using special tokens and smart contracts.

It's like having an online lottery where the rules are written in computer code, and everyone plays fairly using digital tokens.

**Installation steps:**

**Prerequisites:**

Remix IDE: Access the Remix Integrated Development Environment (IDE) here **(**<https://remix.ethereum.org/>).

Solidity Compiler: Ensure the Solidity compiler is available within Remix IDE.

To set up and deploy a Lottery Smart Contract using Solidity programming language in Remix IDE, follow these steps:

**Steps:**

1. **Open Remix IDE:**

Navigate to the [Remix IDE website] (https://remix.ethereum.org/) in your web browser.

1. **Create a New File:**

* Click on the "+" button on the left-hand side panel to create a new file.
* Name the file ‘Lottery.sol’ or any desired name for your contract.

1. **Write the Smart Contract Code:**

* Write the Solidity code for the Lottery Smart Contract in the created file.
* Ensure to define necessary variables, mappings, functions, and constructor (if needed).
* In the newly created file, write your Solidity code for the voting smart contract.

-Here's a basic example:

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.11;

/\*\*

\* @title Storage

\* @dev Store & retrieve value in a variable

\* @custom:dev-run-script ./scripts/deploy\_with\_ethers.ts

\*/

contract Lottery {

address public owner;

address payable[] public players;

uint public lotteryId;

mapping (uint => address payable) public lotteryHistory;

constructor() {

owner = msg.sender;

lotteryId = 1;

}

function getWinnerByLottery(uint lottery) public view returns (address payable) {

return lotteryHistory[lottery];

}

function getBalance() public view returns (uint) {

return address(this).balance;

}

function getPlayers() public view returns (address payable[] memory) {

return players;

}

function enter() public payable {

require(msg.value > 1 ether);

// address of player entering lottery

players.push(payable(msg.sender));

}

function getRandomNumber() public view returns (uint) {

return uint(keccak256(abi.encodePacked(owner, block.timestamp)));

}

function pickWinner() public onlyowner {

uint index = getRandomNumber() % players.length;

players[index].transfer(address(this).balance);

lotteryHistory[lotteryId] = players[index];

lotteryId++;

players = new address payable[](0);

}

modifier onlyowner() {

require(msg.sender == owner);

\_;

}

}

1. **Compile the Contract:**

* Go to the "Solidity Compiler" tab in Remix IDE.
* Select the version of Solidity used in your contract (e.g., `0.8.0`).
* Click on the "Compile Lottery.sol" button. Ensure there are no compilation errors.

1. **Deploy & Run Transactions:**

- Click on "Deploy" to deploy the contract. This action:

* Sets the first account as the manager (1st account 🡪 Enter value 0🡪 Ether🡪 Deploy).
* Selects another accountand sets a value of 2 Ether for participation (Select 2nd account🡪 Enter value 2 🡪 Deployed Contract 🡪Click enter).
* Andrepeat this process for at least 3 participants to create a transaction pool for the lottery.
* Remix will display transaction details, logs, gas usage, and transaction hash for each action executed.

1. **Interacting with the Lottery:**

* As the manager, observe the transaction pool balance and successfully executed transactions.
* To execute the lottery, click on the "Pick Winner" button.

-This action:Randomly selects a winner among all participants.

-Logs the lottery execution details in the console.

1. **Transfer Prize to Winner:**

* After the successful lottery execution, the pool price gets transferred to the winner's account.
* Observe the address (e.g., 0xd9145CCE52D386f254917e481eB44e9943F39138) where the prize is sent.

1. **Generating UML Diagram:**

* Utilize UML (Unified Modelling Language) tools or software to create an architectural diagram.
* Include components like the smart contract, participants, manager, transactions, and winner selection.
* Illustrate the design and implementation of your Lottery DApp using UML diagrams for clarity and documentation purposes.
* For creating a UML diagram:

-Right click on ‘Lottery.sol’ file.

-then click on Generate UML.

**Conclusion:**

Following these steps will enable you to create, deploy, interact with, and visualize your decentralized Lottery DApp using Remix IDE and Ethereum's functionality.

Thus, the build a decentralized Lottery application (DApp), Using ERC-20 Tokens that combine Ethereum's Web3 and Solidity smart contracts deploy on Test network is performed successfully**.**