**🔹 Secure Blockchain Voting System – Step-by-Step Implementation Guide**

We will create a **blockchain-based voting system** using Ethereum, Solidity, Truffle, Ganache, MetaMask, and a front-end in HTML + JavaScript.

**🛠 Step 1: Install Required Tools**

**1️⃣ Install Node.js and npm**

* Download and install [Node.js](https://nodejs.org/) (comes with npm).
* Verify installation:
* node -v
* npm -v

**2️⃣ Install Truffle Framework**

* Open **Command Prompt/Terminal** and run:
* npm install -g truffle
* truffle version

**3️⃣ Install Ganache (Local Ethereum Blockchain)**

* Download **Ganache** from [here](https://trufflesuite.com/ganache/).
* Install and **open** it.
* Click **Quickstart Ethereum** (this starts a local blockchain with test accounts).

**4️⃣ Install MetaMask (Ethereum Wallet)**

* Install **MetaMask** extension from [MetaMask.io](https://metamask.io/).
* Create a **new wallet** (or import using a Ganache private key).
* **Connect MetaMask to Ganache**:
  + Open **MetaMask > Settings > Networks**
  + Click **Add Network > Add Manually**
  + Enter:
    - **Network Name:** Ganache
    - **New RPC URL:** http://127.0.0.1:7545
    - **Chain ID:** 1337
    - Click **Save**

**📁 Step 2: Create Project Folder and Initialize Truffle**

1️⃣ **Open Terminal/Command Prompt**  
2️⃣ **Run the following commands:**

mkdir blockchain-voting

cd blockchain-voting

truffle init

3️⃣ **Check project structure:**

blockchain-voting/

├── contracts/ # Smart contract files

├── migrations/ # Deployment scripts

├── test/ # Test scripts

├── truffle-config.js # Truffle configuration

**📜 Step 3: Configure Truffle to Connect to Ganache**

1️⃣ Open **truffle-config.js** (inside blockchain-voting folder).  
2️⃣ Replace everything with this:

module.exports = {

networks: {

development: {

host: "127.0.0.1",

port: 7545, // Ganache default port

network\_id: "\*", // Connect to any network

},

},

compilers: {

solc: {

version: "0.8.0", // Solidity version

},

},

};

**✍️ Step 4: Write the Smart Contract**

1️⃣ **Create a smart contract file**

cd contracts

touch VotingSystem.sol

2️⃣ **Open VotingSystem.sol and add the following code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract VotingSystem {

address public owner;

struct Voter {

bool isRegistered;

bool hasVoted;

}

mapping(address => Voter) public voters;

mapping(address => uint256) public candidateVotes;

address[] public candidateList;

event VoterRegistered(address voter);

event VoteCast(address voter, address candidate);

modifier onlyOwner() {

require(msg.sender == owner, "Only owner can call this function.");

\_;

}

modifier onlyRegistered() {

require(voters[msg.sender].isRegistered, "Not registered.");

\_;

}

constructor(address[] memory \_candidates) {

owner = msg.sender;

candidateList = \_candidates;

for (uint i = 0; i < candidateList.length; i++) {

candidateVotes[candidateList[i]] = 0;

}

}

function registerVoter(address \_voter) public onlyOwner {

require(!voters[\_voter].isRegistered, "Already registered.");

voters[\_voter] = Voter({ isRegistered: true, hasVoted: false });

emit VoterRegistered(\_voter);

}

function vote(address \_candidate) public onlyRegistered {

require(!voters[msg.sender].hasVoted, "Already voted.");

require(isValidCandidate(\_candidate), "Invalid candidate.");

voters[msg.sender].hasVoted = true;

candidateVotes[\_candidate] += 1;

emit VoteCast(msg.sender, \_candidate);

}

function isValidCandidate(address \_candidate) internal view returns (bool) {

for (uint i = 0; i < candidateList.length; i++) {

if (candidateList[i] == \_candidate) {

return true;

}

}

return false;

}

function getCandidateVotes(address \_candidate) public view returns (uint256) {

require(isValidCandidate(\_candidate), "Invalid candidate.");

return candidateVotes[\_candidate];

}

function getCandidates() public view returns (address[] memory) {

return candidateList;

}

}

**🚀 Step 5: Deploy the Smart Contract**

1️⃣ **Create a deployment file**

cd migrations

touch 2\_deploy\_voting.js

2️⃣ **Open 2\_deploy\_voting.js and add:**

const VotingSystem = artifacts.require("VotingSystem");

module.exports = function (deployer) {

const candidates = [

"0x5FbDB2315678afecb367f032d93F642f64180aa3",

"0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2"

];

deployer.deploy(VotingSystem, candidates);

};

3️⃣ **Compile and Deploy:**

truffle compile

truffle migrate --network development

**🌐 Step 6: Create Front-End for Voting**

1️⃣ **Create a new file index.html in blockchain-voting folder**  
2️⃣ **Paste this code:**

<!DOCTYPE html>

<html>

<head>

<title>Voting DApp</title>

<script src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.min.js"></script>

</head>

<body>

<h1>Secure Blockchain Voting System</h1>

<button id="connect">Connect MetaMask</button>

<button id="vote">Vote</button>

<script>

let provider, signer, contract;

const abi = [...]; // Replace with ABI from truffle compilation

const contractAddress = "YOUR\_DEPLOYED\_CONTRACT\_ADDRESS"; // Replace with actual deployed contract address

document.getElementById("connect").addEventListener("click", async () => {

if (window.ethereum) {

await window.ethereum.request({ method: "eth\_requestAccounts" });

provider = new ethers.providers.Web3Provider(window.ethereum);

signer = provider.getSigner();

contract = new ethers.Contract(contractAddress, abi, signer);

alert("Connected to MetaMask!");

}

});

document.getElementById("vote").addEventListener("click", async () => {

let tx = await contract.vote("CANDIDATE\_ADDRESS\_HERE");

await tx.wait();

alert("Vote cast successfully!");

});

</script>

</body>

</html>

**🎯 Final Step: Running Everything**

1️⃣ Start **Ganache**  
2️⃣ Deploy contract:

truffle migrate --network development

3️⃣ Open index.html in your **browser** and test voting! ✅

**Project Completed Successfully! 🎉🚀**

This guide includes **each and every small step**—just follow them carefully, and you'll complete your project perfectly. If you face any issues, just ask! 😃