Rating.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.14;

contract ReviewSystem {
  struct Candidate {
    uint id;
    string name;
    uint totalRating;
    uint ratingCount;
  }

  address public admin;
  uint public candidatesCount;
  mapping(uint => Candidate) public candidates;
  mapping(address => bool) public hasRated;
  uint public totalRatings;

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

  event CandidateAdded(uint id, string name);
  event Rated(address indexed rater, uint candidateId, uint rating);

  constructor(address _admin) {
    require(_admin != address(0), "Admin address cannot be zero");
    admin = _admin; // Set the initial admin to the specified address
```

```
}

function addCandidate(string memory _name) public onlyAdmin {
    require(bytes(_name).length > 0, "Candidate name cannot be empty");
    candidatesCount++;
    candidates[candidatesCount] = Candidate(candidatesCount, _name, 0, 0);
    emit CandidateAdded(candidatesCount, _name);
}

function rate(uint _candidateId, uint _rating) public {
    require(!hasRated[msg.sender], "You have already rated");
    require(_candidateId > 0 && _candidateId <= candidatesCount, "Invalid candidate ID");
    require(_rating >= 1 && _rating <= 5, "Rating must be between 1 and 5");

    hasRated[msg.sender] = true;
    candidates[_candidateId].totalRating += _rating;
    candidates[_candidateId].ratingCount++;
    totalRatings++;

    emit Rated(msg.sender, _candidateId, _rating);
}

function getAverageRating(uint _candidateId) public view returns (uint) {
    require(_candidateId > 0 && _candidateId <= candidatesCount, "Invalid candidate ID");
    Candidate memory candidate = candidates[_candidateId];
    if (candidate.ratingCount == 0) {
        return 0;
    }
    return candidate.totalRating / candidate.ratingCount;
}
```

```
    function getTotalRatings() public view returns (uint) {
        return totalRatings;
    }

    function changeAdmin(address _newAdmin) public onlyAdmin {
        require(_newAdmin != address(0), "Invalid admin address");
        admin = _newAdmin;
    }
}
```

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Review System</title>
  <script src="https://cdn.jsdelivr.net/npm/web3/dist/web3.min.js"></script>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Arial', sans-serif;
      background-color: #f7f7f7;
      color: #333;
      display: flex;
```

```
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  padding: 20px;
}

h1 {
  font-size: 2.5rem;
  color: #3c3c3c;
  margin-bottom: 40px;
  text-align: center;
}

.btn {
  background-color: #4CAF50;
  color: white;
  padding: 12px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 1rem;
  margin-bottom: 20px;
  transition: all 0.3s ease;
}

.btn:hover {
  background-color: #45a049;
  transform: translateY(-2px);
}
```

```
.section {
  width: 100%;
  max-width: 800px;
  background-color: white;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  padding: 20px;
  margin-top: 30px;
  text-align: center;
}

.input-group {
  margin-bottom: 20px;
}

.input-group input, .input-group select {
  width: 100%;
  padding: 12px;
  border-radius: 5px;
  border: 1px solid #ddd;
  font-size: 1rem;
  margin-top: 10px;
  transition: border 0.3s ease;
}

.input-group input:focus, .input-group select:focus {
  border-color: #4CAF50;
  outline: none;
}

.result-item {
```

```
  display: flex;
  justify-content: space-between;
  margin: 10px 0;
  padding: 10px;
  background-color: #f4f4f4;
  border-radius: 5px;
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.result-item p {
  margin: 0;
}

#role {
  font-weight: bold;
  color: #4CAF50;
  margin-top: 10px;
}

#totalRatings {
  font-weight: bold;
  color: #4CAF50;
  margin-top: 20px;
}

#connectButton {
  font-size: 1.1rem;
  padding: 14px 24px;
  background-color: #007bff;
  border: none;
  color: white;
```

```
    border-radius: 5px;
    cursor: pointer;
    transition: all 0.3s ease;
   }

   #connectButton:hover {
    background-color: #0056b3;
    transform: translateY(-2px);
   }
  </style>
</head>
<body>

  <h1>AUTHENTIC REVIEW SYSTEM</h1>

  <button id="connectButton" class="btn">Connect Wallet</button>
  <p id="account">Not connected</p>
  <p id="role">Role: Not determined</p>

  <!-- Main Section -->
  <div id="mainSection" class="section">
   <h2>Candidate Management</h2>
   <div class="input-group">
    <input type="text" id="candidateName" placeholder="Enter candidate name">
   </div>
   <button id="addCandidateButton" class="btn">Add Candidate</button>
   <h3>Candidate List</h3>
   <div id="candidateList"></div>
  </div>

  <script src="script.js"></script>
```

```
</body>
</html>

<style>
 body {
    background-image: url('team_bg.jpg');
    background-size: cover; /* Ensures the image covers the entire background */
    background-repeat: no-repeat; /* Prevents the image from repeating */
    background-position: center; /* Centers the image */
 }
</style>

<!DOCTYPE html>
<html lang="en">
<head>
  <title>Wallet Connection</title>
  <script src="https://cdn.jsdelivr.net/npm/web3/dist/web3.min.js"></script>
  <style>
    .btn {
      padding: 10px 20px;
      background-color: #007bff;
      color: white;
      border: none;
      cursor: pointer;
      font-size: 16px;
    }
    .btn:hover {
      background-color: #0056b3;
    }
  </style>
</head>
```

```
<body>
  <script>

    const adminAddress = "0x751E50E7340E2dCfbB8623e168785b34eB72579c"; // Replace with actual admin address

    // References to HTML elements
    const connectButton = document.getElementById("connectButton");
    const accountDisplay = document.getElementById("account");
    const roleDisplay = document.getElementById("role");

    // Function to check if MetaMask is installed
    async function checkMetaMask() {
      if (typeof window.ethereum !== "undefined") {
        console.log("MetaMask is installed!");
        return true;
      } else {
        alert("MetaMask is not installed. Please install MetaMask to use this feature.");
        return false;
      }
    }

    // Function to connect wallet and determine role
    async function connectWallet() {
      if (!(await checkMetaMask())) return;

      try {
        // Request account access
        const accounts = await ethereum.request({ method: "eth_requestAccounts" });
        const account = accounts[0];
        accountDisplay.textContent = `Connected: ${account}`;
```

```
            // Determine role based on address
            if (account.toLowerCase() === adminAddress.toLowerCase()) {
                roleDisplay.textContent = "Role: Admin";
            } else {
                roleDisplay.textContent = "Role: User";
            }
        } catch (error) {
            console.error("Error connecting wallet:", error);
            accountDisplay.textContent = "Connection failed";
        }
    }

    // Attach event listener to the button
    connectButton.addEventListener("click", connectWallet);
  </script>
</body>
</html>
```

---

script.js

```
const contractAddress = " "; // contract address
const contractABI =[] //paste your abi code;
//end of abi code
let web3;
let contract;
let account;

// Initialize Web3 and contract
async function initialize() {
  if (window.ethereum) {
```

```
    web3 = new Web3(window.ethereum);
    await window.ethereum.request({ method: "eth_requestAccounts" });
    account = (await web3.eth.getAccounts())[0];
    document.getElementById("account").innerText = `Connected: ${account}`;

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

    loadCandidates();
    listenForAccountChange();
  } else {
    alert("Please install MetaMask to use this dApp!");
  }
}

// Listen for account change
function listenForAccountChange() {
  window.ethereum.on("accountsChanged", (accounts) => {
    account = accounts[0];
    document.getElementById("account").innerText = `Connected: ${account}`;
  });
}

// Add candidate
async function addCandidate() {
  const candidateName = document.getElementById("candidateName").value.trim();
  if (!candidateName) {
    alert("Candidate name cannot be empty!");
    return;
  }

  try {
```

```
    await contract.methods
      .addCandidate(candidateName)
      .send({ from: account });
    alert(`Candidate "${candidateName}" added successfully!`);
    document.getElementById("candidateName").value = ""; // Clear input field
    loadCandidates(); // Refresh candidate list
  } catch (error) {
    console.error(error);
    alert("You are not an admin!");
  }
}

// Load candidates
async function loadCandidates() {
  try {
    const candidatesCount = await contract.methods.candidatesCount().call();
    const candidateList = document.getElementById("candidateList");
    candidateList.innerHTML = "";

    for (let i = 1; i <= candidatesCount; i++) {
      const candidate = await contract.methods.candidates(i).call();
      const averageRating = await contract.methods.getAverageRating(i).call();
      const candidateItem = document.createElement("div");
      candidateItem.className = "result-item";

      candidateItem.innerHTML = `
        <p>${candidate.name} (Avg Rating: ${averageRating})</p>
        <div>
          <input type="number" id="rating-${candidate.id}" min="1" max="5" placeholder="Rate 1-
5">
          <button class="btn" onclick="rateCandidate(${candidate.id})">Rate</button>
```

```
      </div>
    `;

    candidateList.appendChild(candidateItem);
   }
  } catch (error) {
   console.error(error);
   alert("Error loading candidates!");
  }
}

// Rate candidate
async function rateCandidate(candidateId) {
 const ratingInput = document.getElementById(`rating-${candidateId}`);
 const rating = parseInt(ratingInput.value);

 if (isNaN(rating) || rating < 1 || rating > 5) {
  alert("Please provide a valid rating between 1 and 5!");
  return;
 }

 try {
  await contract.methods.rate(candidateId, rating).send({ from: account });
  alert(`Successfully rated candidate with ID ${candidateId}`);
  loadCandidates(); // Refresh candidate list with updated ratings
 } catch (error) {
  console.error(error);
  alert("User can rate a candidate only once");
 }
}
```

```
// Event listeners
document.getElementById("connectButton").addEventListener("click", initialize);
document.getElementById("addCandidateButton").addEventListener("click", addCandidate);
```