Fake Product Detection System

Detection.sol

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

contract fakedet {
  address public admin;

  struct Product {
    string name;
    string manufacturer;
    bool isAuthentic;
    bool exists;
  }

  mapping(string => Product) public products;
  string[] public productIds;

  event ProductAdded(string productId, string name, string manufacturer);
  event ProductVerified(string productId, bool isAuthentic);

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

  constructor() {
    admin = msg.sender;
  }
```

```
    function addProduct(
        string memory productId,
        string memory name,
        string memory manufacturer
    ) public onlyAdmin {
        require(!products[productId].exists, "Product already exists");

        products[productId] = Product({
            name: name,
            manufacturer: manufacturer,
            isAuthentic: true,
            exists: true
        });

        productIds.push(productId);

        emit ProductAdded(productId, name, manufacturer);
    }

    function verifyProduct(string memory productId) public view returns (bool) {
        return products[productId].isAuthentic;
    }

    function getProductCount() public view returns (uint) {
        return productIds.length;
    }

    function getProductByIndex(uint index) public view returns (string memory, string memory,
string memory) {
        require(index < productIds.length, "Invalid index");
        string memory productId = productIds[index];
```

```
        Product memory product = products[productId];
        return (productId, product.name, product.manufacturer);
    }
}
```

Home.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Fake Product Identification</title>
    <script src="https://cdn.jsdelivr.net/npm/web3/dist/web3.min.js"></script>
    <style>
        body { font-family: Arial, sans-serif; max-width: 800px; margin: 0 auto; padding: 20px; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        table, th, td { border: 1px solid #ddd; padding: 8px; }
        th { background-color: #f2f2f2; }
        input { margin: 5px; padding: 5px; width: 200px; }
        button { padding: 5px 10px; margin: 5px; }
    </style>
</head>
<body>
    <h1>Fake Product Identification System</h1>
    <div>
        <h2>Add Product</h2>
        <input type="text" id="productId" placeholder="Product ID">
        <input type="text" id="productName" placeholder="Product Name">
        <input type="text" id="manufacturer" placeholder="Manufacturer">
        <button onclick="addProduct()">Add Product</button>
        <p id="addProductError" style="color: red;"></p>
    </div>
    <div>
```

```
    <h2>Verify Product</h2>
    <input type="text" id="verifyProductId" placeholder="Product ID">
    <button onclick="verifyProduct()">Verify Product</button>
    <p id="verifyResult"></p>
  </div>
  <div>
    <h2>Added Products</h2>
    <button onclick="loadProducts()">Refresh Product List</button>
    <table id="productTable">
      <thead>
        <tr>
          <th>Product ID</th>
          <th>Name</th>
          <th>Manufacturer</th>
        </tr>
      </thead>
      <tbody id="productTableBody">
      </tbody>
    </table>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

Script.js

```
const contractAddress = "" //add deployed address;
const contractABI = [];//add abi code;

let web3;
let contract;
let adminAddress;
```

```
window.addEventListener("load", async () => {
  if (window.ethereum) {
    web3 = new Web3(window.ethereum);
    await ethereum.request({ method: "eth_requestAccounts" });
    contract = new web3.eth.Contract(contractABI, contractAddress);

    // Get admin address
    adminAddress = await contract.methods.admin().call();

    // Load products on initial load
    await loadProducts();
  } else {
    alert("MetaMask is not installed");
  }
});

async function addProduct() {
  const productId = document.getElementById("productId").value;
  const name = document.getElementById("productName").value;
  const manufacturer = document.getElementById("manufacturer").value;
  const addProductError = document.getElementById("addProductError");

  try {
    const accounts = await web3.eth.getAccounts();

    // Check if current account is admin
    if (accounts[0].toLowerCase() !== adminAddress.toLowerCase()) {
      addProductError.textContent = "Only admin can add products";
      return;
    }
```

```
        // Clear previous error
        addProductError.textContent = "";

        // Send transaction
        await contract.methods.addProduct(productId, name, manufacturer).send({ from:
accounts[0] });

        // Refresh product list
        await loadProducts();

        // Clear input fields
        document.getElementById("productId").value = '';
        document.getElementById("productName").value = '';
        document.getElementById("manufacturer").value = '';

        alert("Product added successfully!");
    } catch (error) {
        addProductError.textContent = error.message;
    }
}

async function loadProducts() {
    const tableBody = document.getElementById("productTableBody");
    tableBody.innerHTML = ''; // Clear existing rows

    try {
        // Get total number of products
        const productCount = await contract.methods.getProductCount().call();

        // Iterate through products and add to table
```

```
      for (let i = 0; i < productCount; i++) {
        const productData = await contract.methods.getProductByIndex(i).call();

        const row = tableBody.insertRow();
        row.insertCell(0).textContent = productData[0]; // Product ID
        row.insertCell(1).textContent = productData[1]; // Name
        row.insertCell(2).textContent = productData[2]; // Manufacturer
      }
    } catch (error) {
      console.error("Error loading products:", error);
    }
}

async function verifyProduct() {
    const productId = document.getElementById("verifyProductId").value;
    const isAuthentic = await contract.methods.verifyProduct(productId).call();
    document.getElementById("verifyResult").innerText = isAuthentic ? "The product is authentic."
: "The product is fake.";
}
```