**PRACTICAL: 9.2**

**AIM:** Create Dapp (Decentralized Application) and link your client-side application with Blockchain network created in practical 5 th and 6 th .

**CODE: Event Certificate**

**Contract.js**

import PatientRecords from './artifacts/contracts/PatientRecords.sol/PatientRecords.json';

const contractAddress = "0x267C19bd0466fEAeb5e9a751FDF2c4183a2Bc65c";

const contractABI = PatientRecords.abi;

export { contractAddress, contractABI };

**Web3.js**

import Web3 from 'web3';

let web3;

if (window.ethereum) {

    web3 = new Web3(window.ethereum);

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

} else if (window.web3) {

    web3 = new Web3(window.web3.currentProvider);

} else {

    alert('Please install MetaMask to use this dApp!');

}

export default web3;

**App.js**

import React, { useState, useEffect } from 'react';

import { ethers, BrowserProvider } from 'ethers';

import { contractAddress, contractABI } from './PatientRecords';

import './App.css';

import hospitalLogo from './assets/hospital-logo.svg';

function App() {

  const [account, setAccount] = useState('');

  const [contract, setContract] = useState(null);

  const [patients, setPatients] = useState([]);

  const [loading, setLoading] = useState(true);

  const [searchId, setSearchId] = useState('');

  const [activeTab, setActiveTab] = useState('add'); *// 'add' or 'view'*

  const [formData, setFormData] = useState({

    patient\_id: '',

    patient\_name: '',

    disease\_type: '',

    doctor\_name: '',

    patient\_contact: ''

  });

  useEffect(() => {

    const init = async () => {

      try {

        if (window.ethereum) {

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

          const provider = new BrowserProvider(window.ethereum);

          const signer = await provider.getSigner();

          const address = await signer.getAddress();

          setAccount(address);

          const patientContract = new ethers.Contract(

            contractAddress,

            contractABI,

            signer

          );

          setContract(patientContract);

          await loadAllPatients(patientContract);

        } else {

          alert('Please install MetaMask to use this dApp!');

        }

      } catch (error) {

        console.error('Error initializing the dApp:', error);

      } finally {

        setLoading(false);

      }

    };

    init();

  }, []);

  const loadAllPatients = async (*contractInstance*) => {

    try {

      const allPatients = await contractInstance.getAllPatients();

      setPatients(allPatients);

    } catch (error) {

      console.error('Error loading patients:', error);

      alert('Error loading patients. Please try again.');

    }

  };

  const handleInputChange = (*e*) => {

    const { name, value } = e.target;

    setFormData(*prev* => ({

      ...prev,

      [name]: value

    }));

  };

  const handleSubmit = async (*e*) => {

    e.preventDefault();

    if (!contract) {

      alert('Please connect to MetaMask first');

      return;

    }

    try {

      setLoading(true);

      const tx = await contract.addPatient(

        formData.patient\_id,

        formData.patient\_name,

        formData.disease\_type,

        formData.doctor\_name,

        formData.patient\_contact

      );

      await tx.wait();

      await loadAllPatients(contract);

      setFormData({

        patient\_id: '',

        patient\_name: '',

        disease\_type: '',

        doctor\_name: '',

        patient\_contact: ''

      });

      alert('Patient added successfully!');

    } catch (error) {

      console.error('Error adding patient:', error);

      alert('Error adding patient: ' + error.message);

    } finally {

      setLoading(false);

    }

  };

  const handleSearch = async () => {

    if (!searchId || !contract) {

      alert('Please enter a patient ID and ensure you are connected to MetaMask');

      return;

    }

    try {

      setLoading(true);

*// Convert searchId to a number to ensure proper format*

      const patientId = parseInt(searchId);

      console.log('Searching for patient with ID:', patientId);

      const patient = await contract.getPatient(patientId);

      console.log('Found patient:', patient);

*// Check if patient exists (assuming 0 or empty values mean no patient)*

      if (patient.patient\_id.toString() === '0' || !patient.patient\_name) {

        setPatients([]);

        alert('No patient found with this ID');

        return;

      }

*// If patient found, update the state*

      setPatients([{

        patient\_id: patient.patient\_id,

        patient\_name: patient.patient\_name,

        disease\_type: patient.disease\_type,

        doctor\_name: patient.doctor\_name,

        patient\_contact: patient.patient\_contact

      }]);

    } catch (error) {

      console.error('Error finding patient:', error);

      setPatients([]);

      if (error.message.includes('revert')) {

        alert('Patient not found with this ID');

      } else {

        alert('Error searching for patient: ' + error.message);

      }

    } finally {

      setLoading(false);

    }

  };

*// Update the search input handler to only allow numbers*

  const **handleSearchInputChange** = (*e*) => {

    const value = *e*.target.value;

*// Only allow positive numbers*

    if (value === '' || /^\d+$/.test(value)) {

      setSearchId(value);

    }

  };

  if (loading) {

    return (

      <div *className*="loading-container">

        <div *className*="loader"></div>

        <p>Loading...</p>

      </div>

    );

  }

  return (

    <div *className*="App">

      <header *className*="App-header">

        <img *src*={hospitalLogo} *alt*="Hospital Logo" *className*="hospital-logo" />

        <h1>Patient Records Management System</h1>

      </header>

      <main *className*="main-content">

        <div *className*="tabs">

          <button

*className*={`tab-btn ${activeTab === 'add' ? 'active' : ''}`}

*onClick*={() => setActiveTab('add')}

          >

            Add Patient

          </button>

          <button

*className*={`tab-btn ${activeTab === 'view' ? 'active' : ''}`}

*onClick*={() => setActiveTab('view')}

          >

            View Records

          </button>

        </div>

        {activeTab === 'add' && (

          <section *className*="add-patient-form">

            <h2>Add New Patient</h2>

            <form *onSubmit*={handleSubmit}>

              <div *className*="form-group">

                <label>Patient ID:</label>

                <input

*type*="number"

*name*="patient\_id"

*value*={formData.patient\_id}

*onChange*={handleInputChange}

*required*

                />

              </div>

              <div *className*="form-group">

                <label>Patient Name:</label>

                <input

*type*="text"

*name*="patient\_name"

*value*={formData.patient\_name}

*onChange*={handleInputChange}

*required*

                />

              </div>

              <div *className*="form-group">

                <label>Disease Type:</label>

                <input

*type*="text"

*name*="disease\_type"

*value*={formData.disease\_type}

*onChange*={handleInputChange}

*required*

                />

              </div>

              <div *className*="form-group">

                <label>Doctor Name:</label>

                <input

*type*="text"

*name*="doctor\_name"

*value*={formData.doctor\_name}

*onChange*={handleInputChange}

*required*

                />

              </div>

              <div *className*="form-group">

                <label>Patient Contact:</label>

                <input

*type*="text"

*name*="patient\_contact"

*value*={formData.patient\_contact}

*onChange*={handleInputChange}

*required*

                />

              </div>

              <button *type*="submit" *className*="submit-btn">Add Patient</button>

            </form>

          </section>

        )}

        {activeTab === 'view' && (

          <section *className*="view-records">

            <div *className*="search-container">

              <input

*type*="number"

*placeholder*="Enter Patient ID to search"

*value*={searchId}

*onChange*={handleSearchInputChange}

*min*="1"

*className*="search-input"

              />

              <button *onClick*={handleSearch} *className*="search-btn">

                Find Patient

              </button>

              <button *onClick*={() => loadAllPatients(contract)} *className*="search-btn">

                View All Patients

              </button>

            </div>

            <div *className*="table-container">

              <table *className*="patients-table">

                <thead>

                  <tr>

                    <th>Patient ID</th>

                    <th>Name</th>

                    <th>Disease</th>

                    <th>Doctor</th>

                    <th>Contact</th>

                  </tr>

                </thead>

                <tbody>

                  {patients.map((*patient*, *index*) => (

                    <tr *key*={index}>

                      <td>{patient.patient\_id.toString()}</td>

                      <td>{patient.patient\_name}</td>

                      <td>{patient.disease\_type}</td>

                      <td>{patient.doctor\_name}</td>

                      <td>{patient.patient\_contact}</td>

                    </tr>

                  ))}

                </tbody>

              </table>

              {patients.length === 0 && (

                <p *className*="no-records">No patient records found</p>

              )}

            </div>

          </section>

        )}

      </main>

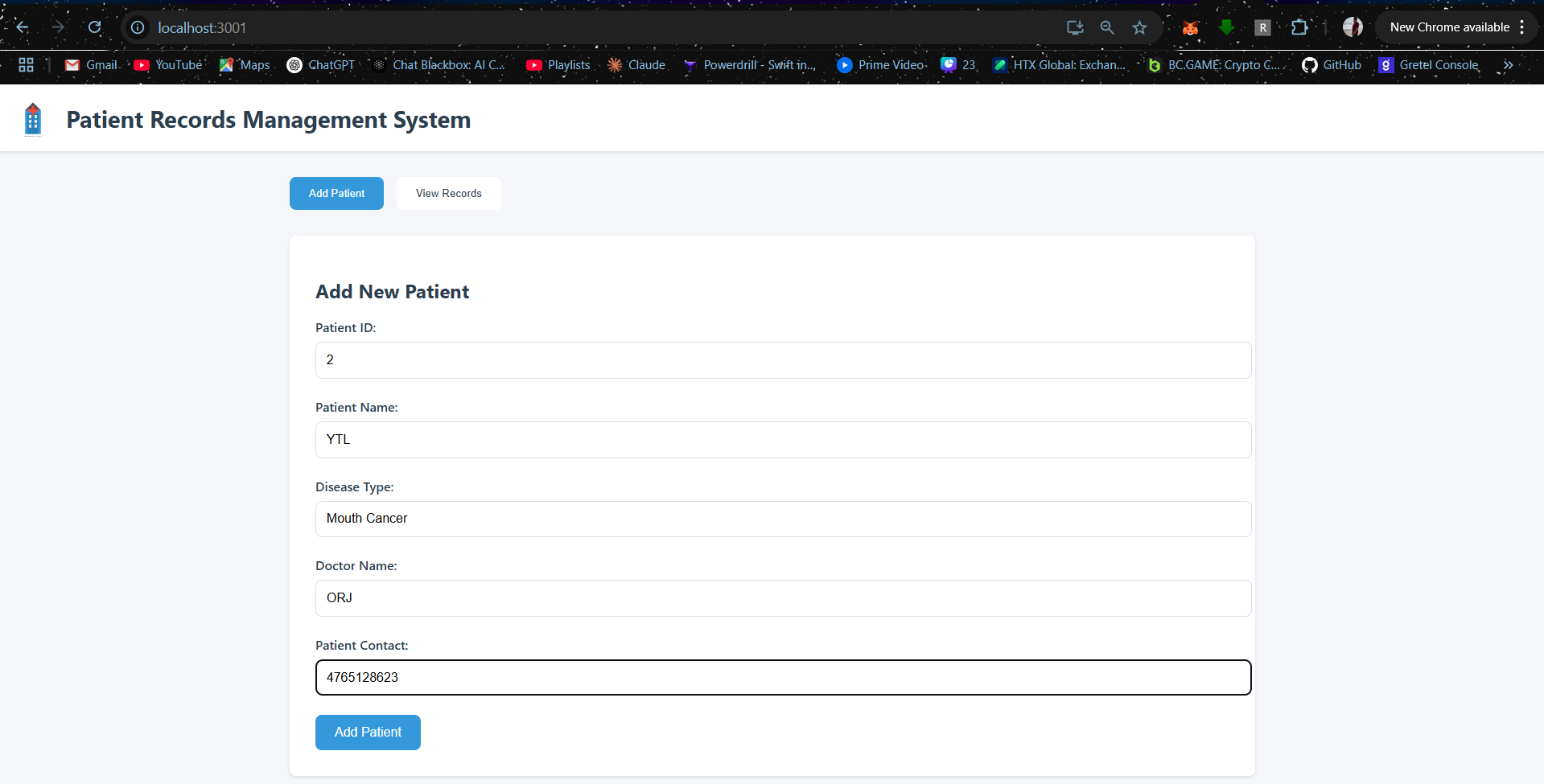
    </div>

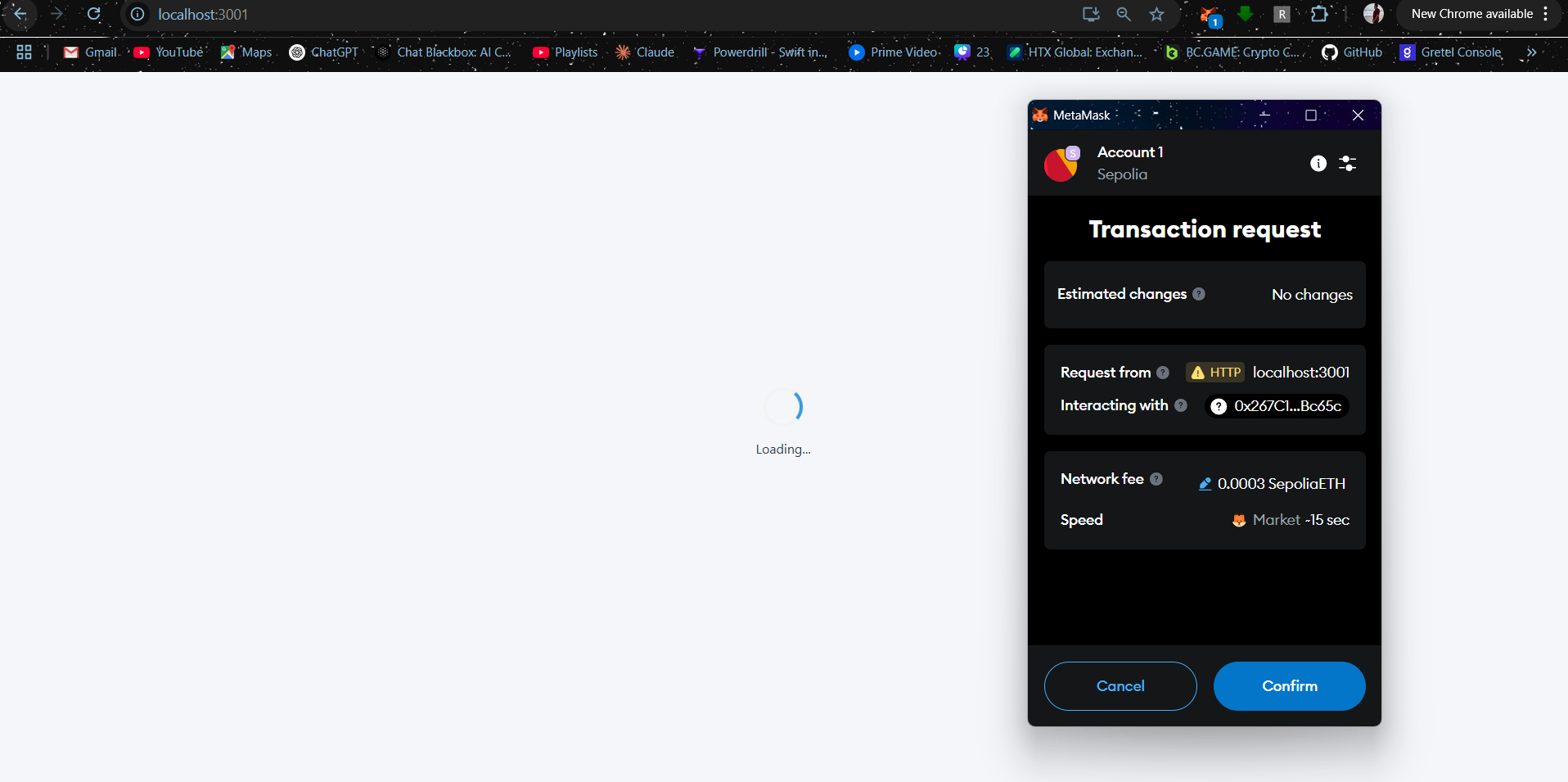
  );

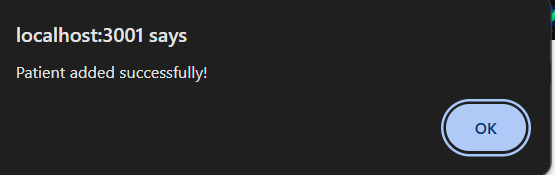
}

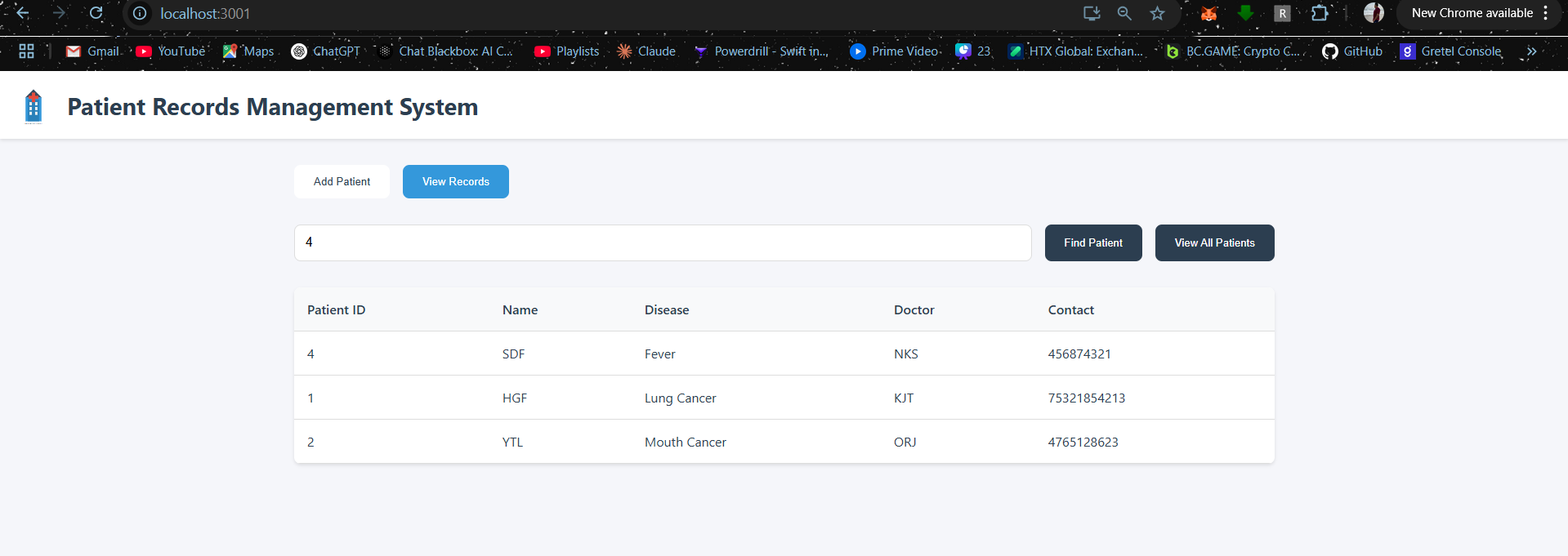
export default App;

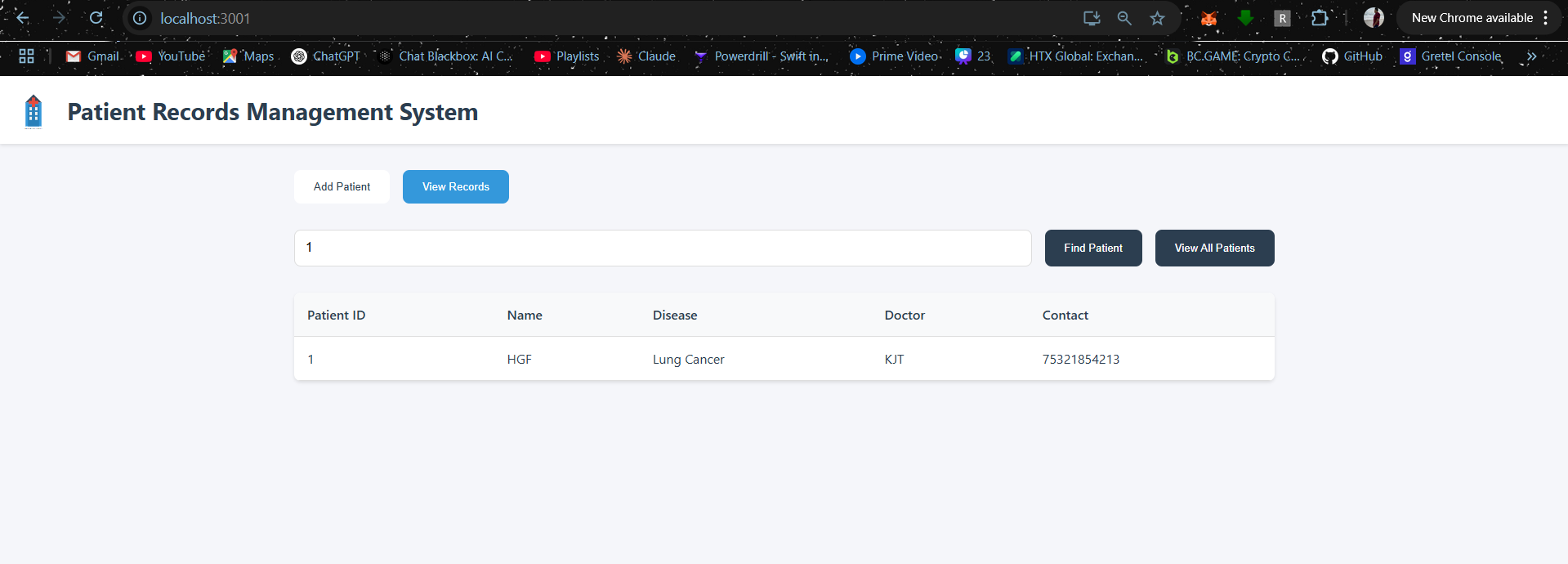
**OUTPUT:**



****

****

****

****

**LEARNING OUTCOME:**

A Decentralized Application (DApp) is built by developing both the client-side application (front-end) and back-end integration with smart contracts. This includes using Metamask for user authentication and performing secure blockchain transactions. The architecture of DApps is studied, understanding how they interact with blockchain networks and smart contracts. Real-world use cases for DApps, such as decentralized finance (DeFi) or identity management, are also explored.