**1.Develop a smart contract for E-voting using assert, revert, and require keyword**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EVoting {

address public owner;

mapping(address => bool) public hasVoted;

mapping(bytes32 => uint256) public votesCount;

event Voted(address indexed voter, bytes32 candidate);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

constructor() {

owner = msg.sender;

}

function vote(bytes32 candidate) external {

require(!hasVoted[msg.sender], "You can only vote once");

hasVoted[msg.sender] = true;

votesCount[candidate]++;

emit Voted(msg.sender, candidate);

}

function getResult(bytes32 candidate) external view returns (uint256) {

return votesCount[candidate];

}

function changeOwner(address newOwner) external onlyOwner {

require(newOwner != address(0), "Invalid new owner address");

owner = newOwner;

}

function selfDestruct() external onlyOwner {

// Use assert to ensure that the contract can only be destroyed by the owner

assert(msg.sender == owner);

selfdestruct(payable(owner));

}

}

**2. Develop a smart contract for storing educational certificates using mapping function get from front end**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract EducationalCertificates {

address public owner;

struct Certificate {

string name;

string course;

uint256 completionYear;

bool isSet;

}

mapping(address => Certificate) public certificates;

event CertificateStored(address indexed recipient, string name, string course, uint256 completionYear);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

constructor() {

owner = msg.sender;

}

function storeCertificate(address recipient, string memory name, string memory course, uint256 completionYear) external onlyOwner {

require(recipient != address(0), "Invalid recipient address");

require(!certificates[recipient].isSet, "Certificate for this address already exists");

certificates[recipient] = Certificate(name, course, completionYear, true);

emit CertificateStored(recipient, name, course, completionYear);

}

function getCertificate(address recipient) external view returns (string memory, string memory, uint256) {

require(certificates[recipient].isSet, "Certificate not found for this address");

Certificate memory cert = certificates[recipient];

return (cert.name, cert.course, cert.completionYear);

}

function changeOwner(address newOwner) external onlyOwner {

require(newOwner != address(0), "Invalid new owner address");

owner = newOwner;

}

}

**3. Develop a smart contract for storing Health Record along with access permission for viewing the record by physicians, lab technicians, and patient.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract HealthRecordSystem {

address public owner;

enum Role {Patient, Physician, LabTechnician}

struct HealthRecord {

string patientName;

string diagnosis;

string testResults;

bool isSet;

}

mapping(address => HealthRecord) public healthRecords;

mapping(Role => mapping(address => bool)) public accessPermissions;

event RecordStored(address indexed patient, string diagnosis, string testResults);

event AccessGranted(address indexed target, Role role);

event AccessRevoked(address indexed target, Role role);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

modifier hasAccess(Role role) {

require(accessPermissions[role][msg.sender], "You do not have access to view this record");

\_;

}

constructor() {

owner = msg.sender;

grantAccess(Role.Patient, msg.sender); // Owner has access to their own record

}

function storeHealthRecord(string memory patientName, string memory diagnosis, string memory testResults) external {

HealthRecord storage record = healthRecords[msg.sender];

record.patientName = patientName;

record.diagnosis = diagnosis;

record.testResults = testResults;

record.isSet = true;

emit RecordStored(msg.sender, diagnosis, testResults);

}

function viewHealthRecord() external view hasAccess(Role.Patient) returns (string memory, string memory, string memory) {

HealthRecord storage record = healthRecords[msg.sender];

return (record.patientName, record.diagnosis, record.testResults);

}

function grantAccess(Role role, address target) external onlyOwner {

accessPermissions[role][target] = true;

emit AccessGranted(target, role);

}

function revokeAccess(Role role, address target) external onlyOwner {

accessPermissions[role][target] = false;

emit AccessRevoked(target, role);

}

function changeOwner(address newOwner) external onlyOwner {

require(newOwner != address(0), "Invalid new owner address");

owner = newOwner;

}

}

**4. Develop a smart contract for supply chain management with following participants**

1. **Provider register for shipping**
2. **Shipment management team**
3. **Tracing**
4. **Customer**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SupplyChainManagement {

address public owner;

enum SupplyChainStage {Registered, Shipped, Received, Completed}

struct Shipment {

address provider;

address shipmentManagement;

address tracing;

address consumer;

string product;

SupplyChainStage stage;

}

mapping(address => Shipment) public shipments;

event ShipmentRegistered(address indexed shipmentId, address indexed provider, string product);

event ShipmentShipped(address indexed shipmentId);

event ShipmentReceived(address indexed shipmentId);

event ShipmentCompleted(address indexed shipmentId);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

modifier onlyProvider(address shipmentId) {

require(shipments[shipmentId].provider == msg.sender, "Only the provider can execute this");

\_;

}

modifier onlyShipmentManagement(address shipmentId) {

require(shipments[shipmentId].shipmentManagement == msg.sender, "Only the shipment management team can execute this");

\_;

}

modifier onlyTracing(address shipmentId) {

require(shipments[shipmentId].tracing == msg.sender, "Only the tracing team can execute this");

\_;

}

modifier onlyConsumer(address shipmentId) {

require(shipments[shipmentId].consumer == msg.sender, "Only the consumer can execute this");

\_;

}

constructor() {

owner = msg.sender;

}

function registerShipment(address shipmentId, string memory product) external {

require(shipments[shipmentId].provider == address(0), "Shipment ID is already registered");

shipments[shipmentId] = Shipment(msg.sender, address(0), address(0), address(0), product, SupplyChainStage.Registered);

emit ShipmentRegistered(shipmentId, msg.sender, product);

}

function assignShipmentManagement(address shipmentId, address shipmentManagement) external onlyProvider(shipmentId) {

shipments[shipmentId].shipmentManagement = shipmentManagement;

}

function shipShipment(address shipmentId) external onlyShipmentManagement(shipmentId) {

shipments[shipmentId].stage = SupplyChainStage.Shipped;

emit ShipmentShipped(shipmentId);

}

function receiveShipment(address shipmentId) external onlyTracing(shipmentId) {

shipments[shipmentId].stage = SupplyChainStage.Received;

emit ShipmentReceived(shipmentId);

}

function completeShipment(address shipmentId) external onlyConsumer(shipmentId) {

shipments[shipmentId].stage = SupplyChainStage.Completed;

emit ShipmentCompleted(shipmentId);

}

function getShipmentDetails(address shipmentId) external view returns (address, address, address, address, string memory, SupplyChainStage) {

Shipment memory shipment = shipments[shipmentId];

return (shipment.provider, shipment.shipmentManagement, shipment.tracing, shipment.consumer, shipment.product, shipment.stage);

}

}

1. **Develop a smart contract for storing educational certificates using mapping function get from front end**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract FinancialTransactions {

address public owner;

mapping(address => uint256) public balances;

event Deposit(address indexed account, uint256 amount);

event Withdrawal(address indexed account, uint256 amount);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

constructor() {

owner = msg.sender;

}

function deposit() external payable {

require(msg.value > 0, "Deposit amount must be greater than 0");

balances[msg.sender] += msg.value;

emit Deposit(msg.sender, msg.value);

}

function withdraw(uint256 amount) external {

require(amount > 0 && amount <= balances[msg.sender], "Invalid withdrawal amount");

balances[msg.sender] -= amount;

payable(msg.sender).transfer(amount);

emit Withdrawal(msg.sender, amount);

}

function getBalance(address account) external view returns (uint256) {

return balances[account];

}

}

Html:-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Financial Transactions DApp</title>

</head>

<body>

<h1>Financial Transactions DApp</h1>

<div>

<h2>Your Balance: <span id="balance">0</span> ETH</h2>

<button onclick="deposit()">Deposit</button>

<button onclick="withdraw()">Withdraw</button>

</div>

<script>

const contractAddress = 'CONTRACT\_ADDRESS'; // Replace with the actual deployed contract address

const contractABI = [ /\* ABI of your smart contract \*/ ];

const web3 = new Web3(window.ethereum);

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

async function updateBalance() {

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

const account = accounts[0];

const balance = await contract.methods.getBalance(account).call();

document.getElementById('balance').innerText = balance;

}

async function deposit() {

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

const account = accounts[0];

const amount = prompt('Enter deposit amount:');

await contract.methods.deposit().send({

from: account,

value: web3.utils.toWei(amount, 'ether'),

});

updateBalance();

}

async function withdraw() {

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

const account = accounts[0];

const amount = prompt('Enter withdrawal amount:');

await contract.methods.withdraw(web3.utils.toWei(amount, 'ether')).send({

from: account,

});

updateBalance();

}

window.onload = async () => {

await window.ethereum.enable();

updateBalance();

};

</script>

</body>

</html>

**6.Develop a smart contract for land registration system with verification of land documents**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract LandRegistrationSystem {

address public owner;

enum VerificationStatus {Pending, Verified, Rejected}

struct LandRegistration {

address ownerAddress;

string landCoordinates;

VerificationStatus status;

}

mapping(uint256 => LandRegistration) public landRegistrations;

uint256 public nextLandId;

event LandRegistered(uint256 indexed landId, address indexed owner, string landCoordinates);

event LandVerified(uint256 indexed landId, VerificationStatus status);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can execute this");

\_;

}

constructor() {

owner = msg.sender;

}

function registerLand(string memory landCoordinates) external {

landRegistrations[nextLandId] = LandRegistration({

ownerAddress: msg.sender,

landCoordinates: landCoordinates,

status: VerificationStatus.Pending

});

emit LandRegistered(nextLandId, msg.sender, landCoordinates);

nextLandId++;

}

function verifyLand(uint256 landId, VerificationStatus status) external onlyOwner {

require(landId < nextLandId, "Invalid land ID");

require(landRegistrations[landId].status == VerificationStatus.Pending, "Land is already verified or rejected");

landRegistrations[landId].status = status;

emit LandVerified(landId, status);

}

function getLandDetails(uint256 landId) external view returns (address, string memory, VerificationStatus) {

require(landId < nextLandId, "Invalid land ID");

LandRegistration memory land = landRegistrations[landId];

return (land.ownerAddress, land.landCoordinates, land.status);

}

}

**7.smart contract for Arithemetic operation**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ArithmeticOperations {

function add(uint256 a, uint256 b) external pure returns (uint256) {

return a + b;

}

function subtract(uint256 a, uint256 b) external pure returns (uint256) {

require(a >= b, "Subtraction result must be non-negative");

return a - b;

}

function multiply(uint256 a, uint256 b) external pure returns (uint256) {

return a \* b;

}

function divide(uint256 a, uint256 b) external pure returns (uint256) {

require(b > 0, "Division by zero");

return a / b;

}

}

**8. Transfer ether from one account to another**

pragma solidity ^0.8.11;

contract EtherTransfer {

uint public amount;

address payable owner;

constructor() {

owner = payable(msg.sender); // set the deployer of contract as the

owner

}

function sendEth(address payable receiver) payable public {

require(owner == msg.sender, "Only the owner can send funds");

amount = msg.value;

receiver.transfer(amount);

}

// Function to check the balance of an account

function checkAccountBalance(address account) external view returns

(uint256) {

return account.balance;

}

}

**9. Smart contract for getter and setter function**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Counter {

uint256 private counter; // Private state variable to store the counter

value

// Getter function to retrieve the current counter value

function getCounter() public view returns (uint256) {

return counter;

}

// Setter function to update the counter value

function setCounter(uint256 \_value) public {

counter = \_value;

}

}

**10. Constructing solidity code for a decentralized application where the owner can create a contract (with a tenant) which can be replicated to all nodes.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract RentalContract {

address public owner;

address public tenant;

uint256 public rentAmount;

bool public contractActive;

event ContractCreated(address indexed \_owner, address indexed \_tenant,

uint256 \_rentAmount);

event ContractTerminated();

modifier onlyOwner() {

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

\_;

}

modifier onlyTenant() {

require(msg.sender == tenant, "Only the tenant can call this

function");

\_;

}

constructor(address \_tenant, uint256 \_rentAmount) {

owner = msg.sender;

tenant = \_tenant;

rentAmount = \_rentAmount;

contractActive = true;

emit ContractCreated(owner, tenant, rentAmount);

}

function terminateContract() public onlyOwner {

contractActive = false;

emit ContractTerminated();

}

function payRent() public payable onlyTenant {

require(contractActive, "Contract is not active");

require(msg.value == rentAmount, "Incorrect rent amount");

}

receive() external payable {

// Handle incoming Ether (e.g., rent payments)

}

}

**11. In a rented house setup with the owner and the tenants, the tenant can submit a deposit and the contract’s state changes on all the decentralized nodes.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract RentalContract {

address public owner;

address public tenant;

uint256 public depositAmount;

bool public isDepositPaid;

event DepositReceived(address indexed \_tenant, uint256 \_amount);

modifier onlyOwner() {

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

\_;

}

modifier onlyTenant() {

require(msg.sender == tenant, "Only the tenant can call this

function");

\_;

}

constructor(address \_tenant, uint256 \_depositAmount) {

owner = msg.sender;

tenant = \_tenant;

depositAmount = \_depositAmount;

isDepositPaid = false;

}

function payDeposit() public payable onlyTenant {

require(!isDepositPaid, "Deposit has already been paid");

require(msg.value == depositAmount, "Incorrect deposit amount");

isDepositPaid = true;

emit DepositReceived(msg.sender, msg.value);

}

function refundDeposit() public onlyOwner {

require(isDepositPaid, "Deposit has not been paid yet");

payable(tenant).transfer(depositAmount);

isDepositPaid = false;

}

}

**12. The owner should be able to check the balance of the contract from any one of the nodes.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract RentalContract {

address public owner;

address public tenant;

uint256 public depositAmount;

bool public isDepositPaid;

event DepositReceived(address indexed \_tenant, uint256 \_amount);

modifier onlyOwner() {

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

\_;

}

modifier onlyTenant() {

require(msg.sender == tenant, "Only the tenant can call this

function");

\_;

}

constructor(address \_tenant, uint256 \_depositAmount) {

owner = msg.sender;

tenant = \_tenant;

depositAmount = \_depositAmount;

isDepositPaid = false;

}

function payDeposit() public payable onlyTenant {

require(!isDepositPaid, "Deposit has already been paid");

require(msg.value == depositAmount, "Incorrect deposit amount");

isDepositPaid = true;

emit DepositReceived(msg.sender, msg.value);

}

function refundDeposit() public onlyOwner {

require(isDepositPaid, "Deposit has not been paid yet");

payable(tenant).transfer(depositAmount);

isDepositPaid = false;

}

// Function to check the contract's balance

function getContractBalance() public view onlyOwner returns (uint256) {

return address(this).balance;

}

receive() external payable {

// Handle incoming Ether (e.g., rent payments)

}

}

**13. Created a contract, with restrictions about withdrawing the funds from certain account, preferably the owners with different levels of security restrictions.**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract RestrictedWithdrawalContract {

address public owner;

uint256 public balance;

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can call this

function.");

\_;

}

constructor() {

owner = msg.sender;

}

function deposit() public payable {

require(msg.value > 0, "You must send some ether to deposit.");

balance += msg.value;

}

function withdraw(uint256 amount) public onlyOwner {

require(amount <= balance, "Insufficient balance.");

balance -= amount;

payable(owner).transfer(amount);

}

function getBalance() public view returns (uint256) {

return balance;

}

}

**14. Create an dapp for storing the medical records of patients**

**Anim.css**

html {

background: #e3e3e3;

}

.container {

display: flex;

align-items: center;

justify-content: center;

}

#redblocks > \* {

transform: translate3d(160px, -93px, 0);

animation: moveblocks 4s 1s ease infinite;

}

@keyframes moveblocks {

0% {

transform: translate3d(160px, -93px, 0);

}

50%,

100% {

transform: translate(0);

}

}

#firstBlock {

transform: translate3d(160px, -93px, 0);

animation: firstBlock 4s 1s ease infinite;

}

#blockdis {

animation: blockdis 4s 1s ease infinite;

}

@keyframes blockdis {

30% {

opacity: 1;

}

40%,

100% {

opacity: 0;

transform: translate3d(-160px, 93px, 0);

}

}

@keyframes firstBlock {

0%,

15% {

opacity: 0;

}

40%,

100% {

opacity: 1;

}

}

@for $i from 1 to 5 {

#redblocksparticles g:nth-child(#{$i}) polygon {

opacity: 0.35;

animation: glow 4s 1s ease infinite;

}

}

@for $i from 1 to 20 {

#redblocksparticles g:nth-child(1) g circle:nth-child(#{$i}) {

animation: dots 4s #{1000 + $i \* 50}ms ease infinite;

}

#redblocksparticles g:nth-child(2) g circle:nth-child(#{$i}) {

animation: dots 4s #{1000 + $i \* 50}ms ease infinite;

}

#redblocksparticles g:nth-child(3) g circle:nth-child(#{$i}) {

animation: dots 4s #{1000 + $i \* 50}ms ease infinite;

}

#redblocksparticles g:nth-child(4) g circle:nth-child(#{$i}) {

animation: dots 4s #{1000 + $i \* 50}ms ease infinite;

}

}

@keyframes glow {

0%,

45% {

opacity: 0;

}

60%,

100% {

opacity: 0.35;

}

}

@keyframes dots {

0%,

35% {

transform: translate(0);

opacity: 0;

}

60%,

80% {

transform: translate(55px, -35px);

opacity: 1;

}

100% {

transform: translate(55px, -35px);

opacity: 0;

}

}

@for $i from 1 to 22 {

#purplebg > :nth-child(#{$i}) {

animation: up 2s #{$i \* 500}ms ease infinite alternate;

}

}

@keyframes up {

to {

transform: translate(0, -25px);

}

}

@for $i from 1 to 13 {

#bottomparticles > :nth-child(#{$i}) {

animation: particles 4s #{$i \* 300}ms ease infinite alternate,

p 2s ease infinite alternate;

}

#bottomparticles2 > :nth-child(#{$i}) {

animation: particles 4s #{$i \* 300}ms ease infinite alternate,

p 2s ease infinite alternate;

}

#redglowparticles > :nth-child(#{$i}) {

animation: particles 4s #{$i \* 300}ms ease infinite alternate,

p 2s ease infinite alternate;

}

}

@keyframes p {

85%,

100% {

opacity: 0;

}

}

@keyframes particles {

0%,

100% {

transform: translate(0);

}

50% {

transform: translate(10px, 15px);

}

}

.particlespoly {

animation: p 2s ease infinite alternate;

}

@for $i from 1 to 4 {

#d-app g:nth-child(#{$i}) {

animation: updown 2s #{$i \* 200}ms ease-in-out infinite alternate;

}

}

@for $i from 1 to 4 {

#d-apps2 g:nth-child(#{$i}) {

animation: updown 2s #{$i \* 200}ms ease-in-out infinite alternate;

}

}

@for $i from 1 to 20 {

#d-apps2wrapper g:nth-child(3) g > circle:nth-child(#{$i}) {

animation: updown 2s #{$i \* 50}ms ease infinite alternate;

}

}

@for $i from 1 to 20 {

#d-appswrapper g:nth-child(2) g > circle:nth-child(#{$i}) {

animation: updown 2s #{$i \* 50}ms ease infinite alternate;

}

}

@keyframes updown {

100% {

transform: translate(0, -20px);

}

}

#Layer\_1 > g:nth-child(2) > g:nth-child(18) > g:nth-child(12) {

animation: arrows 1s ease-in-out infinite alternate,

p 2s ease infinite alternate;

}

@keyframes arrows {

to {

transform: translate(25px, 25px);

}

}

#Layer\_1 > g:nth-child(2) > g:nth-child(17) > g:nth-child(13) {

animation: arrows2 1s ease-in-out infinite alternate,

p 2s ease infinite alternate;

}

@keyframes arrows2 {

to {

transform: translate(-25px, -25px);

}

}

#Layer\_1 > g:nth-child(2) > g:nth-child(17) > g:nth-child(12) {

animation: arrows 1s ease-in-out infinite alternate,

p 2s ease infinite alternate;

}

**Indexhtml**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<meta name="author" content="">

<title>SwasthyaChain</title>

<!-- Bootstrap Core CSS -->

<link href="css/bootstrap.min.css" rel="stylesheet">

<link rel="icon" type="image/x-icon" href="/favicon.ico"/>

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

<!-- Custom CSS -->

<style>

.panel-heading{

margin-bottom: 20px;

}

.navbar{

margin-bottom: 70px;

}

</style>

</head>

<body>

<!-- Navigation -->

<nav class="navbar navbar-inverse navbar-static-top" role="navigation">

<div class="container-fluid">

<!-- Brand and toggle get grouped for better mobile display -->

<div class="navbar-header">

<button type="button" class="navbar-toggle" datatoggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">SwasthyaChain</a>

</div>

<!-- Collect the nav links, forms, and other content for toggling

-->

<div class="collapse navbar-collapse" id="bs-example-navbarcollapse-1">

<ul class="nav navbar-nav navbar-right">

<li class="active">

<a href="./index.html">Login</a>

</li>

<li>

<a href="./register.html">Register</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container -->

</nav>

<div><div id="includedContent"></div></div>

<!-- Page Content -->

<div class="container">

<div class="panel panel-default">

<div class="panel-heading"><h3 class="text-center">Welcome to

SwasthyaChain</h3></div>

<div class="panel-body">

<div class="row">

<div class="alert alert-warning col-sm-8 col-sm-offset-2"

style="display: none;">

<strong>Warning!</strong> Unregistered user. Click <a

href="./register.html">here</a> to register.

</div>

</div>

<form class="form-horizontal" name="loginForm">

<div class="panel-heading"><h5 class="text-center">Login

is Linked to your Metamask Account</h5></div>

<div class="form-group">

</div>

</form>

<div class="text-center">

<button class="btn btn-primary btn-lg"

onclick="login()">Login</button>

</div>

</div>

<div class="panel-footer">

</div>

</div>

</div>

<!-- /.container -->

<!-- jQuery Version 1.11.1 -->

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

<!-- Bootstrap Core JavaScript -->

<script src="js/bootstrap.min.js"></script>

<script src="/js/web3.min.js"></script>

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

<script>

$(function(){

$("#includedContent").load("anim.html");

});

connect();

function login(){

$(".alert-warning").hide();

publicKey= web3.currentProvider.selectedAddress;

console.log(publicKey);

contractInstance.get\_patient\_list(function(error, result){

if(!error){

var PatientList = result;

for(var i = 0; i < PatientList.length; i++) {

if (publicKey.toLowerCase() == PatientList[i]) {

location.href = "./patient.html?key=" + publicKey;

}

}

} else {

console.log(error);

console.log("Invalid User!");

$(".alert-warning").show();

}

});

contractInstance.get\_doctor\_list(function(error, result){

if(!error){

var DoctorList = result;

for(var i = 0; i < DoctorList.length; i++) {

if (publicKey.toLowerCase() == DoctorList[i]) {

location.href = "./doctor.html?key=" + publicKey;

}

}

} else {

console.log(error);

$(".alert-warning").show();

}

});

console.log("Invalid User!");

$(".alert-warning").show();

}

</script>

</body>

</html>

**Patient.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<meta name="author" content="">

<title>SwasthyaChain</title>

<!-- Bootstrap Core CSS -->

<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- <script src="js/bundle.js"></script> -->

<!-- Custom CSS -->

<style>

body {

/\* padding-top: 70px; \*/

/\* Required padding for .navbar-fixed-top. Remove if using .navbarstatic-top. Change if height of navigation changes. \*/

}

.navbar{

margin-bottom: 70px;

}

.panel-heading{

margin-bottom: 20px;

}

.panel{

margin-bottom: 60px;

}

/\* .publicKeyDoctor{

display: none;

} \*/

.well > h3{

margin: 10px auto;

}

.checkbox label:after {

content: '';

display: table;

clear: both;

}

.checkbox .cr {

position: relative;

display: inline-block;

border: 1px solid #a9a9a9;

border-radius: .25em;

width: 1.3em;

height: 1.3em;

float: left;

margin-right: .5em;

}

.checkbox .cr .cr-icon {

position: absolute;

font-size: .8em;

line-height: 0;

top: 50%;

left: 15%;

}

.checkbox label input[type="checkbox"] {

display: none;

}

.checkbox label input[type="checkbox"]+.cr>.cr-icon {

opacity: 0;

}

.checkbox label input[type="checkbox"]:checked+.cr>.cr-icon {

opacity: 1;

}

</style>

</head>

<body>

<!-- Navigation -->

<nav class="navbar navbar-inverse navbar-static-top"

role="navigation">

<div class="container-fluid">

<!-- Brand and toggle get grouped for better mobile display --

>

<div class="navbar-header">

<button type="button" class="navbar-toggle" datatoggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">SwasthyaChain</a>

</div>

<!-- Collect the nav links, forms, and other content for

toggling -->

<div class="collapse navbar-collapse" id="bs-example-navbarcollapse-1">

<ul class="nav navbar-nav navbar-right">

<li>

<a href="./index.html">Logout</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container -->

</nav>

<div class="container">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="text-center">Personal Information</h3>

</div>

<div class="panel-body">

<div class="row">

<div class="col-sm-offset-1 col-sm-10">

<table class="table">

<tr>

<th>Name:</th>

<td id="name"></td>

</tr>

<tr>

<th>Age:</th>

<td id="age"></td>

</tr>

</table>

<div class="text-center">

<h5>Your records are stored here:

http://localhost:8080/ipfs/<span id="recordsHash"></span></h5>

<button type="submit" class="btn btn-info btnlg" onclick="showRecords(this)">View medical records</button>

</div>

<pre id="records" style="margin-top: 20px;">

</pre>

</div>

</div>

</div>

</div>

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="text-center">Share your Medical Record</h3>

</div>

<div class="panel-body">

<div class="row">

<div class="alert alert-info col-sm-8 col-sm-offset2">

<strong>Info!</strong> The doctor already has

access to your records. Revoke access or wait for time duration to end to

continue.

</div>

</div>

<form class="form-horizontal" action="/action\_page.php">

<div class="form-group">

<label for="permitDoctorList" class="control-label

col-sm-2">Doctor:</label>

<div class="col-sm-8">

<select class="form-control"

id="permitDoctorList">

<option selected disabled>-- Please Select

--</option>

</select>

</div>

</div>

</form>

<div class="text-center">

<button onclick = "giveAccess()" class="btn btnprimary btn-lg">Submit</button>

</div>

</div>

</div>

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="text-center">Current EMR access holders</h3>

</div>

<div class="panel-body">

<div class="row">

<div class="alert alert-danger col-sm-8 col-sm-offset2">

<strong>Notice!</strong> The access could not be

revoked. Please retry or contact admin.

</div>

</div>

<div class="row">

<div class="col-sm-offset-1 col-sm-10">

<table id="accessDoc" class="table table-hover">

<tr>

<th>Doctor</th>

<th class="publicKeyDoctor">Public

Key</th>

<th>Revoke access</th>

</tr>

</table>

</div>

</div>

</div>

</div>

</div>

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

<!-- Bootstrap Core JavaScript -->

<script src="js/bootstrap.min.js"></script>

<script src="/js/web3.min.js"></script>

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

<script>

var url\_string = window.location.href;

var url = new URL(url\_string);

var key ;

toggleRecordsButton = 0;

var recordHash = "";

$(window).load(function() {

connect();

$("#records").hide();

$(".alert-info").hide();

$(".alert-danger").hide();

key = web3.currentProvider.selectedAddress;

key = key.toLocaleLowerCase();

var a = "";

var b = 0;

var ailments = [];

var insurerName = "";

$("#buyInsurance").hide();

$("#insuranceInfo").hide();

// print patient details and insurer details (if exists). If insurer

does not exist show the buy insurance panel

console.log("Getting Patient Data");

contractInstance.get\_patient.call(key, {gas: 1000000}, function(error,

result){

console.log("Patient Data Result:"+result);

if(!error){

console.log(result);

a = result[0];

b = result[1];

ailments = result[2];

insurerAddress = result[3];

recordHash = result[4];

$("#name").html(a);

$("#age").html(b.c[0]);

$("#recordsHash").html(recordHash);

if (insurerAddress != 0){

$("#buyInsurance").hide();

} else {

var InsurerList = 0;

contractInstance.get\_insurer\_list({gas: 1000000},

function(error, result){

if(!error){

InsurerList = result;

var list = document.getElementById("insurers");

for (var i = 0; i < InsurerList.length; i++) {

contractInstance.get\_insurer.call(InsurerList[

i], {gas: 1000000}, function(error, result){

if(!error){

d = result[0];

var option =

document.createElement("option");

option.text = d;

list.add(option);

} else{

console.log(error);

}

})

}

}

})

$("#buyInsurance").show();

$("#insuranceInfo").hide();

}

}

else

console.error(error);

});

// print out the doctors to share emr

var DoctorList = 0;

console.log("Getting Doctor List");

contractInstance.get\_doctor\_list({gas: 1000000},function(error,

result){

if(!error) {

DoctorList = result;

for(var i = 0; i < DoctorList.length; i++) {

contractInstance.get\_doctor.call(DoctorList[i], {gas:

1000000},function(error, result){

var list =

document.getElementById("permitDoctorList");

if(!error) {

[a, b] = result;

var option = document.createElement("option");

option.text = a;

list.add(option);

// console.log(a);

}

else

console.error(error);

})

}

}

else

console.error(error);

});

// print out the doctors who have access

var doctorAddressList = 0;

contractInstance.get\_accessed\_doctorlist\_for\_patient(key, {gas:

1000000}, function(error, result){

if(!error){

doctorAddressList = result;

// console.log(result);

doctorAddressList.forEach(function(doctorAddress, index){

contractInstance.get\_doctor.call(doctorAddress, {gas:

1000000}, function(error, result){

var table = document.getElementById("accessDoc");

if(!error) {

[a,b] = result;

console.log(a);

var row = table.insertRow(index+1);

var cell1 = row.insertCell(0);

var cell2 = row.insertCell(1);

var cell3 = row.insertCell(2);

cell2.className = "publicKeyDoctor";

cell1.innerHTML = a;

cell2.innerHTML = doctorAddress;

cell3.innerHTML = '<button

onclick="revokeAccess(this)" class="btn btn-danger">Revoke access</button>';

console.log(result);

}

else

console.error(error);

})

})

}

else

console.error(error);

});

});

function showRecords(element){

if (toggleRecordsButton%2 == 0){

$.get("http://localhost:8080/ipfs/"+recordHash, function(data){

$("#records").html(data);

$("#records").show();

})

toggleRecordsButton +=1

element.innerHTML = "Hide Medical Records";

element.className = "btn btn-info btn-lg";

} else{

$("#records").hide();

toggleRecordsButton -=1;

element.innerHTML = "View Medical Records";

element.className = "btn btn-info btn-lg"

}

}

function giveAccess() {

var list = document.getElementById("permitDoctorList");

index = list.selectedIndex;

var DoctorList = 0;

contractInstance.get\_doctor\_list({gas: 1000000}, function(error,

result){

if(!error) {

// console.log(index);

DoctorList = result;

doctorToBeAdded = DoctorList[index-1];

contractInstance.permit\_access.sendTransaction(doctorToBeAdded

, {from: key, gas: 1000000, value: web3.toWei(2, 'ether')}, function(error){

if (!error) {

var table = document.getElementById("accessDoc");

noRows = table.rows.length;

var row = table.insertRow(noRows);

var cell1 = row.insertCell(0);

var cell2 = row.insertCell(1);

var cell3 = row.insertCell(2);

cell2.className = "publicKeyDoctor";

cell1.innerHTML = $("#permitDoctorList").val();

cell2.innerHTML = doctorToBeAdded;

cell3.innerHTML = '<button

onclick="revokeAccess(this)" class="btn btn-danger">Revoke access</button>';

} else {

$(".alert-info").show();

console.log(error);

}

})

} else

console.log(error);

})

}

function revokeAccess(element) {

rowNo = element.parentNode.parentNode.rowIndex;

Row = element.parentNode.parentNode;

var Cells = Row.getElementsByTagName("td");

var docKey = Row.cells[1].firstChild.nodeValue;

contractInstance.revoke\_access(docKey, {gas: 1000000},

function(error){

if (!error) {

document.getElementById("accessDoc").deleteRow(rowNo);

} else {

$(".alert-danger").show();

console.log(error);

}

});

}

</script>

</body>

</html>

**Register.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="description" content="">

<meta name="author" content="">

<title>SwasthyaChain</title>

<!-- Bootstrap Core CSS -->

<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- Custom CSS -->

<style>

.panel-heading{

margin-bottom: 20px;

}

.navbar{

margin-bottom: 70px;

}

</style>

<script src="/js/web3.min.js"></script>

</head>

<body>

<!-- Navigation -->

<nav class="navbar navbar-inverse navbar-static-top"

role="navigation">

<div class="container-fluid">

<!-- Brand and toggle get grouped for better mobile

display -->

<div class="navbar-header">

<button type="button" class="navbar-toggle" datatoggle="collapse" data-target="#bs-example-navbar-collapse-1">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">SwasthyaChain</a>

</div>

<!-- Collect the nav links, forms, and other content for

toggling -->

<div class="collapse navbar-collapse" id="bs-examplenavbar-collapse-1">

<ul class="nav navbar-nav navbar-right">

<li>

<a href="./index.html">Login</a>

</li>

<li class="active">

<a href="./register.html">Register</a>

</li>

</ul>

</div>

<!-- /.navbar-collapse -->

</div>

<!-- /.container -->

</nav>

<div class="container">

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="text-center">Please enter your details to

register.</h3>

</div>

<div class="panel-body">

<div class="row">

<div class="alert alert-warning col-sm-8 col-smoffset-2" style="display: none;">

<strong>Warning!</strong> Invalid public key.

Please enter a valid public key to register.

</div>

<div class="alert alert-info col-sm-8 col-sm-offset-2"

style="display: none;">

<strong>Info!</strong> User already registered.

Please check your details.

</div>

</div>

<form name="registerForm" class="form-horizontal"

action="./doctor.html">

<div class="form-group">

<label class="control-label col-sm-2"

for="name">Name:</label>

<div class="col-sm-8">

<input type="text" class="form-control"

id="name" placeholder="Enter name" name = "Name" required autofocus>

</div>

</div>

<div class="form-group">

<label class="control-label col-sm-2"

for="age">Age:</label>

<div class="col-sm-8">

<input type="age" class="form-control"

id="age" placeholder="Enter age" name = "Age" required>

</div>

</div>

<div class="form-group">

<label for="designation" class="control-label colsm-2">Registering as</label>

<div class="col-sm-8">

<select class="form-control" id="designation"

required>

<option selected disabled>-- Please Select

--</option>

<option value="0">Patient</option>

<option value="1">Doctor</option>

</select>

</div>

</div>

</form>

<div class="text-center">

<button class="btn btn-primary btn-lg"

onclick="addAgent()">Register</button>

</div>

</div>

</div>

<hr>

</div>

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

<!-- Bootstrap Core JavaScript -->

<script src="js/bootstrap.min.js"></script>

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

<script src="/js/web3.min.js"></script>

<script src="https://unpkg.com/ipfs-api/dist/index.min.js"

crossorigin="anonymous"></script>

<script>

function addAgent(){

var ipfs = window.IpfsApi('localhost', '5001')

const Buffer = window.IpfsApi().Buffer;

name = $("#name").val();

age = $("#age").val();

designation = $("#designation").val();

designation = parseInt(designation);

publicKey = web3.currentProvider.selectedAddress;

publicKey = publicKey.toLowerCase();

console.log("PK:"+publicKey);

var validPublicKey = true;

var validAgent = true;

var PatientList = 0;

var DoctorList = 0;

var InsurerList = 0;

contractInstance.get\_patient\_list({gas: 1000000},function(error,

result){

if(!error)

PatientList = result;

else

console.error(error);

});

contractInstance.get\_doctor\_list({gas: 1000000},function(error,

result){

if(!error)

DoctorList = result;

else

console.error(error);

});

contractInstance.get\_insurer\_list({gas: 1000000},function(error,

result){

if(!error)

InsurerList = result;

else

console.error(error);

});

if (validPublicKey == false) {

$(".alert-warning").show();

}

else{

for(j = 0; j < PatientList.length; j++) {

if (publicKey == PatientList[j] ){

validAgent = false;

}

}

for(j = 0; j < DoctorList.length; j++) {

if (publicKey == DoctorList[j] ){

validAgent = false;

}

}

for(j = 0; j < InsurerList.length; j++) {

if (publicKey == InsurerList[j] ){

validAgent = false;

}

}

console.log(validAgent);

if (validAgent == true) {

$(".alert-warning").hide()

$(".alert-info").hide();

var ipfshash = "";

if (designation == "0") {

var reportTitle =

`Name: ${name}

Public Key: ${publicKey}

`;

var buffer = Buffer(reportTitle);

ipfs.files.add(buffer, (error, result) => {

if(error){

console.log(error);

}else{

console.log("result:"+result);

ipfshash = result[0].hash;

console.log("Ipfs hash:"+ipfshash);

contractInstance.add\_agent(name, age, designation,

ipfshash, {gas: 1000000}, (err, res) => {

if(!err){

location.replace("./patient.html");

}else{

console.log(err);

}

})

}

})

}else{

contractInstance.add\_agent(name, age, designation,

ipfshash, {gas: 1000000}, (err, res) => {

if (!err) {

if (designation == "1") {

location.replace("./doctor.html");

}

} else

console.log(err);

});

}

}

else {

$(".alert-info").show();

}

}

return false;

}

</script>

</body>

</html>

**Agent.sol**

pragma solidity ^0.5.1;

contract Agent {

struct patient {

string name;

uint age;

address[] doctorAccessList;

uint[] diagnosis;

string record;

}

struct doctor {

string name;

uint age;

address[] patientAccessList;

}

uint creditPool;

address[] public patientList;

address[] public doctorList;

mapping (address => patient) patientInfo;

mapping (address => doctor) doctorInfo;

mapping (address => address) Empty;

// might not be necessary

mapping (address => string) patientRecords;

function add\_agent(string memory \_name, uint \_age, uint \_designation, string

memory \_hash) public returns(string memory){

address addr = msg.sender;

if(\_designation == 0){

patient memory p;

p.name = \_name;

p.age = \_age;

p.record = \_hash;

patientInfo[msg.sender] = p;

patientList.push(addr)-1;

return \_name;

}

else if (\_designation == 1){

doctorInfo[addr].name = \_name;

doctorInfo[addr].age = \_age;

doctorList.push(addr)-1;

return \_name;

}

else{

revert();

}

}

function get\_patient(address addr) view public returns (string memory , uint, uint[]

memory , address, string memory ){

// if(keccak256(patientInfo[addr].name) == keccak256(""))revert();

return (patientInfo[addr].name, patientInfo[addr].age,

patientInfo[addr].diagnosis, Empty[addr], patientInfo[addr].record);

}

function get\_doctor(address addr) view public returns (string memory , uint){

// if(keccak256(doctorInfo[addr].name)==keccak256(""))revert();

return (doctorInfo[addr].name, doctorInfo[addr].age);

}

function get\_patient\_doctor\_name(address paddr, address daddr) view public

returns (string memory , string memory ){

return (patientInfo[paddr].name,doctorInfo[daddr].name);

}

function permit\_access(address addr) payable public {

require(msg.value == 2 ether);

creditPool += 2;

doctorInfo[addr].patientAccessList.push(msg.sender)-1;

patientInfo[msg.sender].doctorAccessList.push(addr)-1;

}

//must be called by doctor

function insurance\_claim(address paddr, uint \_diagnosis, string memory \_hash)

public {

bool patientFound = false;

for(uint i = 0;i<doctorInfo[msg.sender].patientAccessList.length;i++){

if(doctorInfo[msg.sender].patientAccessList[i]==paddr){

msg.sender.transfer(2 ether);

creditPool -= 2;

patientFound = true;

}

}

if(patientFound==true){

set\_hash(paddr, \_hash);

remove\_patient(paddr, msg.sender);

}else {

revert();

}

bool DiagnosisFound = false;

for(uint j = 0; j < patientInfo[paddr].diagnosis.length;j++){

if(patientInfo[paddr].diagnosis[j] == \_diagnosis)DiagnosisFound = true;

}

}

function remove\_element\_in\_array(address[] storage Array, address addr) internal

returns(uint)

{

bool check = false;

uint del\_index = 0;

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

if(Array[i] == addr){

check = true;

del\_index = i;

}

}

if(!check) revert();

else{

if(Array.length == 1){

delete Array[del\_index];

}

else {

Array[del\_index] = Array[Array.length - 1];

delete Array[Array.length - 1];

}

Array.length--;

}

}

function remove\_patient(address paddr, address daddr) public {

remove\_element\_in\_array(doctorInfo[daddr].patientAccessList, paddr);

remove\_element\_in\_array(patientInfo[paddr].doctorAccessList, daddr);

}

function get\_accessed\_doctorlist\_for\_patient(address addr) public view returns

(address[] memory )

{

address[] storage doctoraddr = patientInfo[addr].doctorAccessList;

return doctoraddr;

}

function get\_accessed\_patientlist\_for\_doctor(address addr) public view returns

(address[] memory )

{

return doctorInfo[addr].patientAccessList;

}

function revoke\_access(address daddr) public payable{

remove\_patient(msg.sender,daddr);

msg.sender.transfer(2 ether);

creditPool -= 2;

}

function get\_patient\_list() public view returns(address[] memory ){

return patientList;

}

function get\_doctor\_list() public view returns(address[] memory ){

return doctorList;

}

function get\_hash(address paddr) public view returns(string memory ){

return patientInfo[paddr].record;

}

function set\_hash(address paddr, string memory \_hash) internal {

patientInfo[paddr].record = \_hash;

}

}