**Experiment no.3 : Designing a Smart Contract for Storing and Displaying Information of Students**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract StudentInformation {

struct Student {

string name;

uint256 rollNo;

uint256 percentage;

string email;

uint256 dob; // Date of Birth as Unix timestamp

string studentClass;

uint256 admissionYear;

}

mapping(address => Student) public students;

function storeStudentData(

string memory \_name,

uint256 \_rollNo,

uint256 \_percentage,

string memory \_email,

uint256 \_dob,

string memory \_studentClass,

uint256 \_admissionYear

) public {

students[msg.sender] = Student({

name: \_name,

rollNo: \_rollNo,

percentage: \_percentage,

email: \_email,

dob: \_dob,

studentClass: \_studentClass,

admissionYear: \_admissionYear

});

}

function displayStudentData() public view returns (

string memory,

uint256,

uint256,

string memory,

uint256,

string memory,

uint256

) {

Student memory student = students[msg.sender];

return (

student.name,

student.rollNo,

student.percentage,

student.email,

student.dob,

student.studentClass,

student.admissionYear

);

}

}); }

**Experiment no.4 : Designing a Smart Contract for Bank Account Management**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract BankAccount {

string public depositorName;

uint256 public accountNumber;

string public accountType;

uint256 public balance;

constructor(string memory \_depositorName, uint256 \_accountNumber, string memory

\_accountType) {

depositorName = \_depositorName;

accountNumber = \_accountNumber;

accountType = \_accountType;

balance = 0;

}

function deposit(uint256 amount) public {

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

balance += amount;

}

function withdraw(uint256 amount) public {

require(amount > 0, "Withdrawal amount must be greater than 0");

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

balance -= amount;

}

function displayBalance() public view returns (uint256) {

return balance;

}

function displayNameAndBalance() public view returns (string memory, uint256) {

return (depositorName, balance);

}

}, balance);

**Experiment no.6 : Designing a Web Application for Bank Account Management**

Step 1: Set Up the Project

1. Open your terminal or command prompt.

2. Create a new directory for your project.

mkdir bank-account-management

cd bank-account-management-account-management

3. Initialize a new Node.js project.

npm init -ynpm init -y

4. Install the necessary dependencies: Express and Body-parser.

npm install express body-parsernpm install express body-parser

Step 2: Create the Web Application

1. Create a new file named app.js.

2. Open app.js in your text editor.

3. Start by requiring the necessary modules.

4. const express = require(&#39;express&#39;);

5. const bodyParser = require(&#39;body-parser&#39;);

6. const app = express();

7. Set up middleware for parsing JSON requests.

app.use(bodyParser.json()); ());

8. Define the state variables and methods for the bank account.

let accounts = [];

class BankAccount {

constructor(name, accountNumber, accountType, balance) {

this.name = name;

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = balance;

}

deposit(amount) {

this.balance += amount;

return this.balance;

}

withdraw(amount) {

if (amount &gt; this.balance) {

return &quot;Insufficient balance&quot;;

}

this.balance -= amount;

return this.balance;

}

display() {

return `Name: ${this.name}, Balance: ${this.balance}`;

}

}} }

9. Implement the API endpoints for creating an account, depositing funds,

withdrawing funds, and displaying account details.

// API endpoints

// Create an account

app.post(&#39;/account&#39;, (req, res) =&gt; {

const { name, accountNumber, accountType, balance } = req.body;

const newAccount = new BankAccount(name, accountNumber, accountType, balance);

accounts.push(newAccount);

res.json({ message: &#39;Account created successfully&#39; });

});

// Deposit to an account

app.post(&#39;/account/:accountNumber/deposit&#39;, (req, res) =&gt; {

const accountNumber = req.params.accountNumber;

const { amount } = req.body;

const account = accounts.find(acc =&gt; acc.accountNumber === accountNumber);

if (!account) {

return res.status(404).json({ message: &#39;Account not found&#39; });

}

account.deposit(amount);

res.json({ message: &#39;Amount deposited successfully&#39;, balance: account.balance });

});

// Withdraw from an account

app.post(&#39;/account/:accountNumber/withdraw&#39;, (req, res) =&gt; {

const accountNumber = req.params.accountNumber;

const { amount } = req.body;

const account = accounts.find(acc =&gt; acc.accountNumber === accountNumber);

if (!account) {

return res.status(404).json({ message: &#39;Account not found&#39; });

}

const remainingBalance = account.withdraw(amount);

if (typeof remainingBalance === &#39;string&#39;) {

return res.status(400).json({ message: remainingBalance });

}

res.json({ message: &#39;Amount withdrawn successfully&#39;, balance: remainingBalance });

});

// Display account details

app.get(&#39;/account/:accountNumber&#39;, (req, res) =&gt; {

const accountNumber = req.params.accountNumber;

const account = accounts.find(acc =&gt; acc.accountNumber === accountNumber);

if (!account) {

return res.status(404).json({ message: &#39;Account not found&#39; });

}

res.json({ name: account.name, balance: account.balance });

});}); });

Step 3: Test the Application

1. Save app.js.

2. Open your terminal and navigate to the project directory.

3. Start the server by running:

node app.jsnode app.js

4. Open Postman or a similar tool to test the API endpoints.

5. Send requests to create accounts, deposit and withdraw funds, and display

account details.

**Experiment no.5:Designing a Web3.js Application for Student Information Management**

Step 1: Write the Smart Contract

1. Open your preferred text editor.

2. Write the Solidity code for the student information smart contract. The contract

should define state variables for student attributes and include methods for

storing and reading student data. Here&#39;s an example:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract StudentInformation {

struct Student {

string name;

uint256 rollNo;

uint256 percentage;

string email;

uint256 dob; // Date of Birth as Unix timestamp

string studentClass;

uint256 admissionYear;

}

mapping(address =&gt; Student) public students;

function storeStudentData(

string memory \_name,

uint256 \_rollNo,

uint256 \_percentage,

string memory \_email,

uint256 \_dob,

string memory \_studentClass,

uint256 \_admissionYear

) public {

students[msg.sender] = Student({

name: \_name,

rollNo: \_rollNo,

percentage: \_percentage,

email: \_email,

dob: \_dob,

studentClass: \_studentClass,

admissionYear: \_admissionYear

});

}

}Step 2: Compile and Deploy the Smart Contract

1. Compile the smart contract using Remix or any other Solidity compiler.

2. Deploy the compiled smart contract to an Ethereum network. Note down the

contract address and ABI (Application Binary Interface).

Step 3: Create the HTML Interface

1. Create a new HTML file (e.g., index.html) in your project directory.

2. Write the HTML code to create a user interface for interacting with the smart

contract. Include input fields to enter student information and buttons to store

and display data. Here&#39;s an example:

&lt;!DOCTYPE html&gt;

&lt;html lang=&quot;en&quot;&gt;

&lt;head&gt;

&lt;meta charset=&quot;UTF-8&quot;&gt;

&lt;meta name=&quot;viewport&quot; content=&quot;width=device-width, initial-scale=1.0&quot;&gt;

&lt;title&gt;Student Information DApp&lt;/title&gt;

&lt;script src=&quot;https://cdn.jsdelivr.net/npm/web3@1.5.3/dist/web3.min.js&quot;&gt;&lt;/script&gt;

&lt;script&gt;

window.addEventListener(&#39;load&#39;, async () =&gt; {

if (window.ethereum) {

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

await window.ethereum.enable();

} else {

console.log(&#39;Web3 not found&#39;);

}

});

async function storeStudentData() {

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

const contractAddress = &#39;CONTRACT\_ADDRESS&#39;; // Replace with the address of

the deployed contract

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

const name = document.getElementById(&#39;name&#39;).value;

const rollNo = document.getElementById(&#39;rollNo&#39;).value;

const percentage = document.getElementById(&#39;percentage&#39;).value;

const email = document.getElementById(&#39;email&#39;).value;

const dob = Date.parse(document.getElementById(&#39;dob&#39;).value);

const studentClass = document.getElementById(&#39;class&#39;).value;

const admissionYear = document.getElementById(&#39;admissionYear&#39;).value;

try {

await contract.methods.storeStudentData(name, rollNo, percentage, email, dob,

studentClass, admissionYear).send({

from: accounts[0]

});

alert(&#39;Student data stored successfully!&#39;);

} catch (error) {

console.error(&#39;Error storing student data:&#39;, error);

}

}

async function displayStudentData() {

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

const contractAddress = &#39;CONTRACT\_ADDRESS&#39;; // Replace with the address of

the deployed contract

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

try {

const student = await contract.methods.students(accounts[0]).call();

alert(&#39;Student Information:\nName: &#39; + student.name + &#39;\nRoll No: &#39; +

student.rollNo + &#39;\nPercentage: &#39; + student.percentage + &#39;\nEmail: &#39; + student.email +

&#39;\nDate of Birth: &#39; + new Date(student.dob \* 1000).toDateString() + &#39;\nClass: &#39; +

student.studentClass + &#39;\nAdmission Year: &#39; + student.admissionYear);

} catch (error) {

console.error(&#39;Error displaying student data:&#39;, error);

}

}

&lt;/script&gt;

&lt;/head&gt;

&lt;body&gt;

&lt;h1&gt;Student Information DApp&lt;/h1&gt;

&lt;label for=&quot;name&quot;&gt;Name:&lt;/label&gt;

&lt;input type=&quot;text&quot; id=&quot;name&quot;&gt;&lt;br&gt;

&lt;label for=&quot;rollNo&quot;&gt;Roll No:&lt;/label&gt;

&lt;input type=&quot;number&quot; id=&quot;rollNo&quot;&gt;&lt;br&gt;

&lt;label for=&quot;percentage&quot;&gt;Percentage:&lt;/label&gt;

&lt;input type=&quot;number&quot; id=&quot;percentage&quot;&gt;&lt;br&gt;

&lt;label for=&quot;email&quot;&gt;Email:&lt;/label&gt;

&lt;input type=&quot;email&quot; id=&quot;email&quot;&gt;&lt;br&gt;

&lt;label for=&quot;dob&quot;&gt;Date of Birth:&lt;/label&gt;

&lt;input type=&quot;date&quot; id=&quot;dob&quot;&gt;&lt;br&gt;

&lt;label for=&quot;class&quot;&gt;Class:&lt;/label&gt;

&lt;input type=&quot;text&quot; id=&quot;class&quot;&gt;&lt;br&gt;

&lt;label for=&quot;admissionYear&quot;&gt;Admission Year:&lt;/label&gt;

&lt;input type=&quot;number&quot; id=&quot;admissionYear&quot;&gt;&lt;br&gt;

&lt;button onclick=&quot;storeStudentData()&quot;&gt;Store Student Data&lt;/button&gt;

&lt;button onclick=&quot;displayStudentData()&quot;&gt;Display Student Data&lt;/button&gt;

&lt;/body&gt;

&lt;/html&gt;button&gt; &lt;/body&gt; &lt;/html&gt;

Replace CONTRACT\_ADDRESS and ABI with the address and ABI of your deployed

smart contract respectively.

Step 4: Testing

1. Open the HTML file in a web browser with MetaMask or another Ethereum

wallet extension installed.

2. Use the input fields to enter student information and click the &quot;Store Student

Data&quot; button to store the data on the blockchain.

3. Click the &quot;Display Student Data&quot; button to retrieve and display the stored

student information.

**Experiment no.2:Writing a Solidity Program to Sum All Prime Numbers from 1 to 300**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract PrimeNumberSum {

function isPrime(uint256 number) internal pure returns (bool) {

if (number &lt;= 1) {

return false;

}

for (uint256 i = 2; i &lt;= number / 2; i++) {

if (number % i == 0) {

return false;

}

}

return true;

}

function sumPrimes() public pure returns (uint256) {

uint256 sum = 0;

for (uint256 i = 2; i &lt;= 300; i++) {

if (isPrime(i)) {

sum += i;

}

}

return sum;

}

}return

**Experiment no.1:Writing a Solidity Program to Reverse Digits of a Positive Integer**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ReverseInteger {

function reverse(uint256 n) public pure returns (uint256) {

require(n &gt; 0, &quot;Input must be a positive integer&quot;);

uint256 reversed = 0;

uint256 remainder;

while (n != 0) {

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n = n / 10;

}

return reversed;

}

})