

Practical No - 4

Aim : Securing Transactions and Wallets

- 1. Secure transaction handling in a blockchain**
- 2. Setting up and securing cryptocurrency wallets**
- 3. Hands-on experience with cryptographic key management**

Requirements : MEW wallet, Remix IDE, Ganache.

Theory:

Managing Keys

1. Key Management Tools:

- a) Use hardware wallets (like Ledger or Trezor) for storing large amounts.
- b) Use MEW or MetaMask for managing keys and making transactions.

2. Security Best Practices:

- a) Always double-check the recipient address before sending transactions.
- b) Enable two-factor authentication (2FA) where possible.
- c) Regularly update your wallet software.

Setting Up a Secure Wallet and Managing Keys:

Step-1:

- Go to the MEW website and create an account to go to MEW website click here.

Step-2:

- Now click on create a new wallet, After Clicking that choose the Enkrypt Browser wallet and create an account.

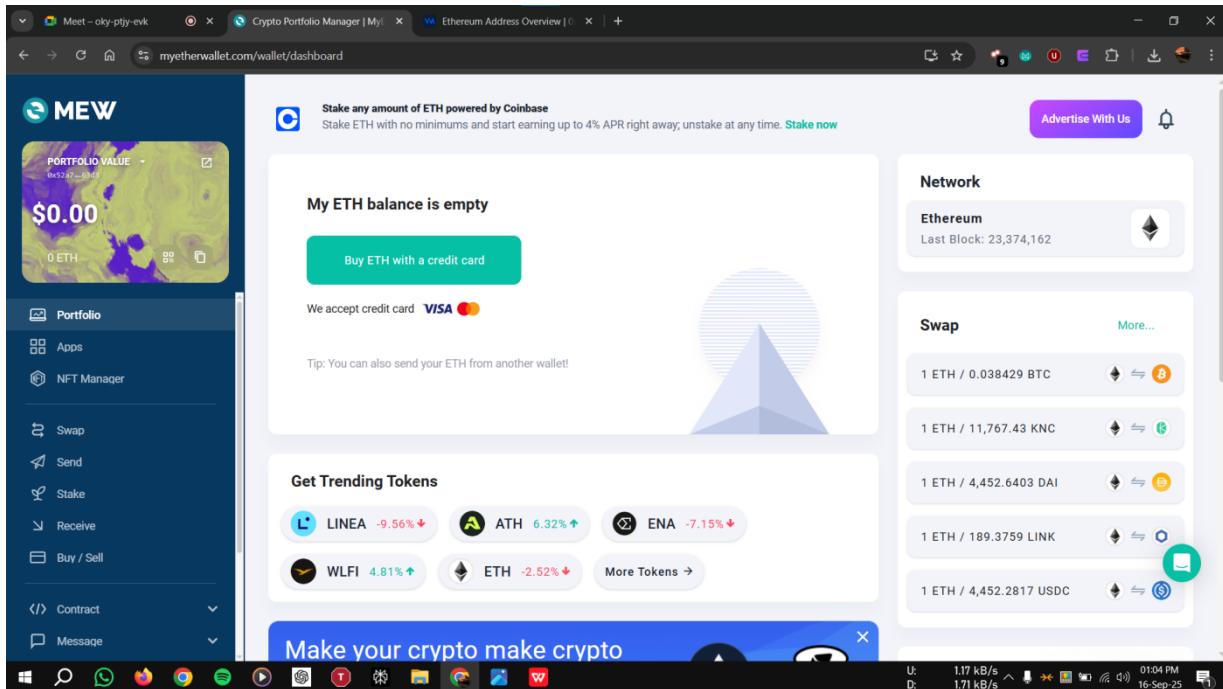


Step-3:

- After clicking that the extension will be downloaded to the browser extension, after that open the extension then we need to create an account in that wallet. And then come back to the MEW website and click on access my wallet.

Step-4:

- After that click on Enkrypt then it will connect to the extension of Enkrypt and it will show the dashboard of the account like following.

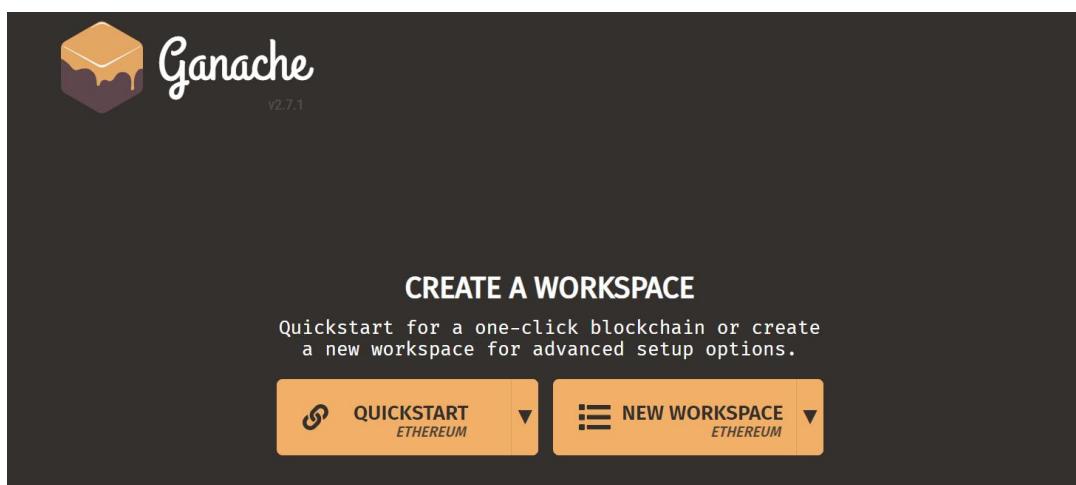


Step-5:

- While creating the account we will get the mnemonic phrase we need to keep that phrase secure and from the access my wallet we can able to download the key store file we need to keep that key store file save for further purpose.

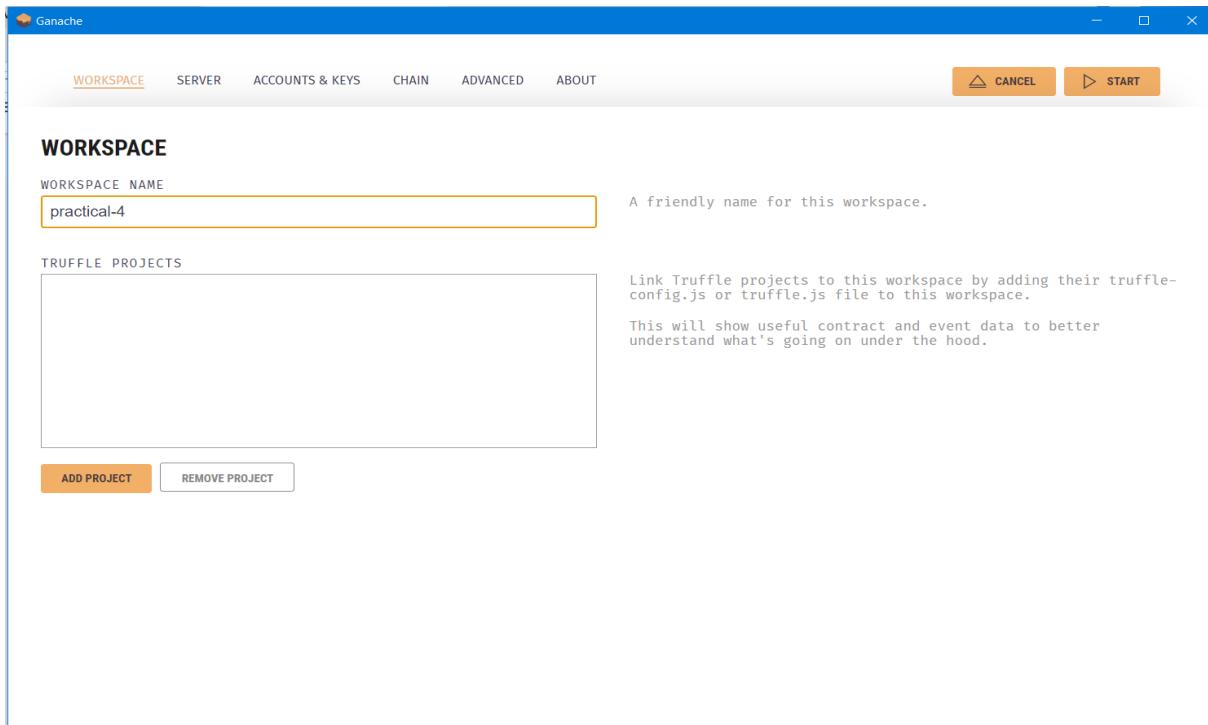
Step-6:

- Now open the ganache and click on new workspace.



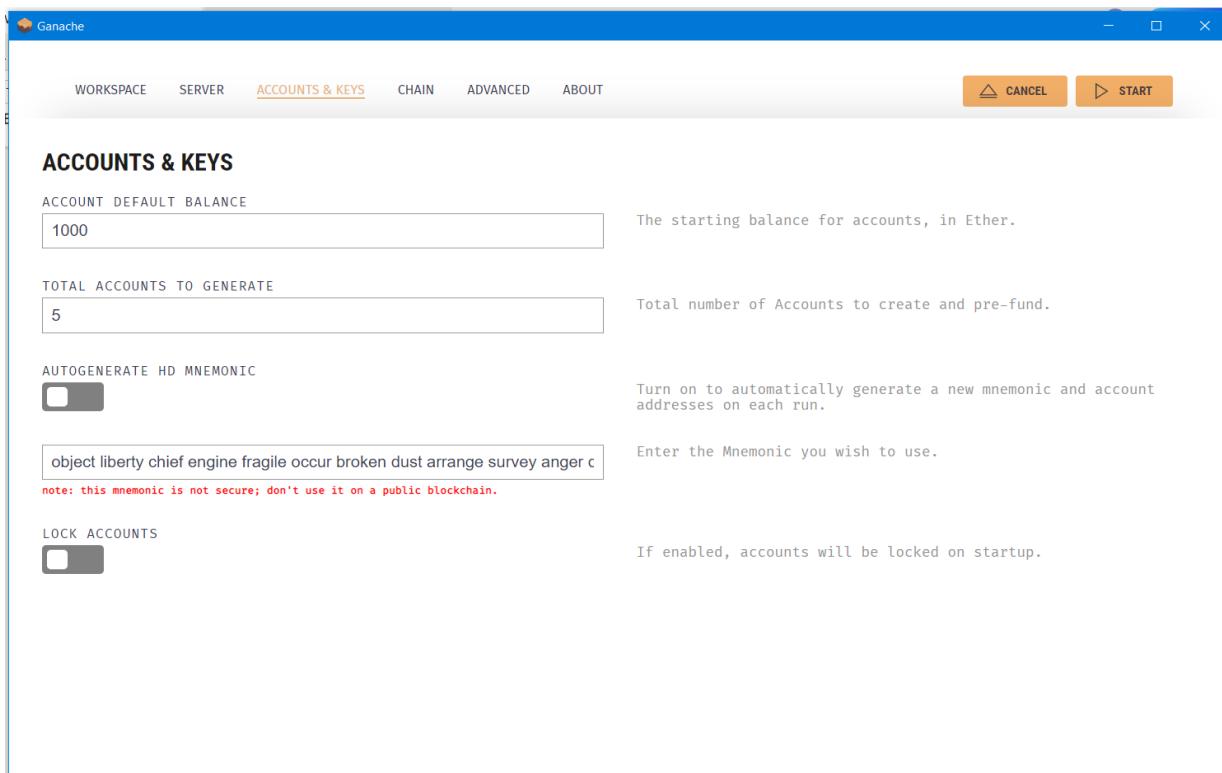
Step-7:

- After clicking on new workspace give the project name as Practical-4 and then click on the Account and Keys.



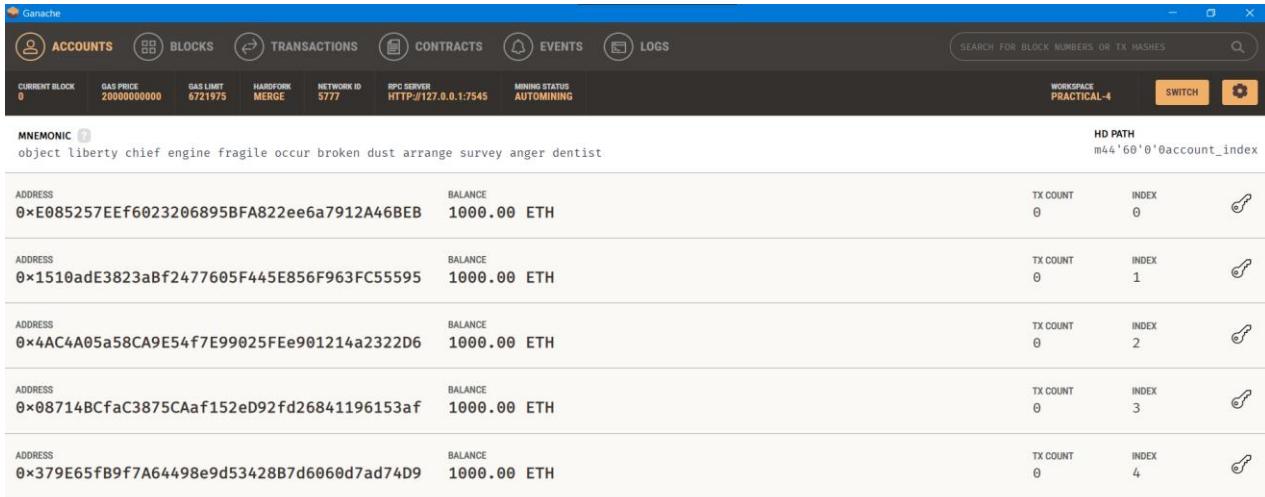
Step-8:

- After we need to go to the accounts and key option in the ganache. Over there we need to change the Accounts default balance to the 1000 and Total accounts to generate to the 5 and then hit on the start button on the right top of the ganache screen.



Step-9:

- After clicking on the start button, you can able to see the 5 accounts with the 1000 test ethers.



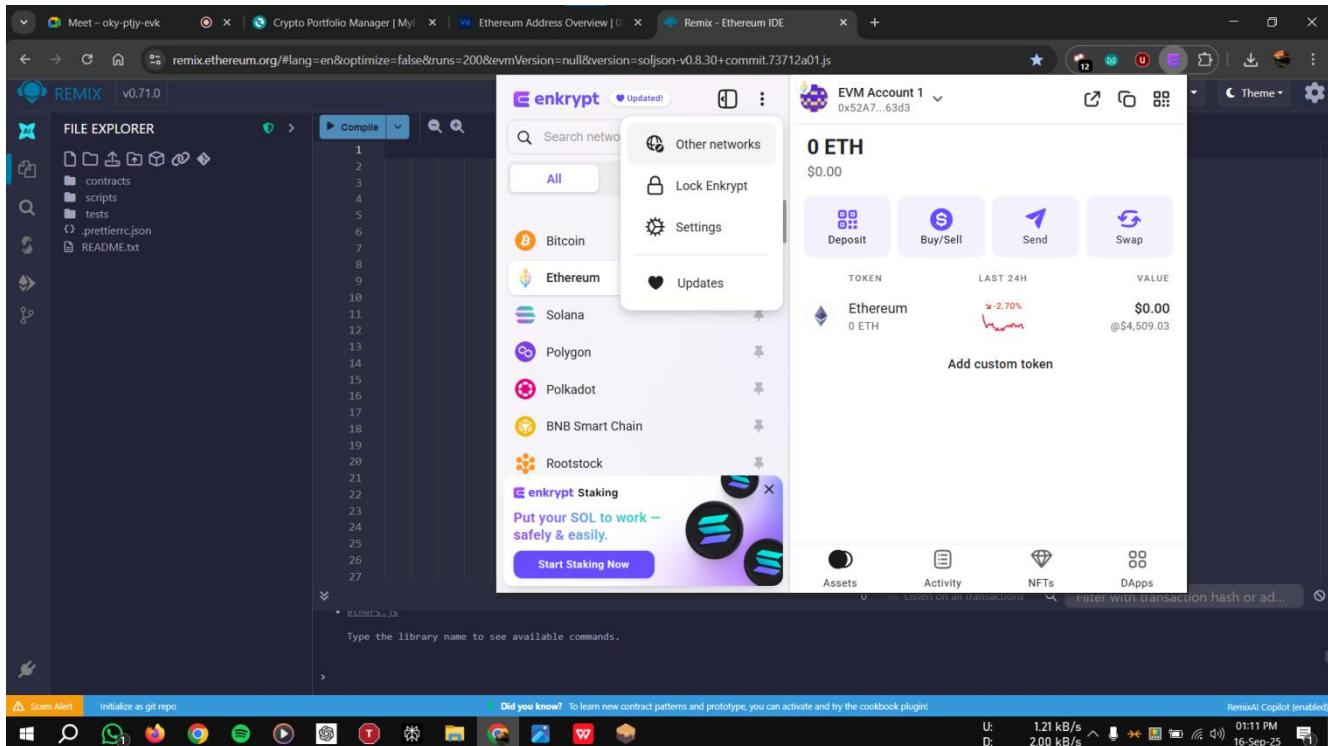
The screenshot shows the Ganache application interface. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below the tabs, there are status indicators for CURRENT BLOCK (0), GAS PRICE (20000000000), GAS LIMIT (6721975), HARDFORK (MERGE), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), and MINING STATUS (AUTOMINING). A search bar at the top right says "SEARCH FOR BLOCK NUMBERS OR TX HASHES". The main area displays five accounts, each with a balance of 1000.00 ETH. The accounts are listed as follows:

ADDRESS	BALANCE	TX COUNT	INDEX	
0xE085257EEf6023206895BFA822ee6a7912A46BEB	1000.00 ETH	0	0	
0x1510adE3823aBf2477605F445E856F963FC55595	1000.00 ETH	0	1	
0x4AC4A05a58CA9E54f7E99025FEe901214a2322D6	1000.00 ETH	0	2	
0x08714BCfaC3875CAaf152eD92fd26841196153af	1000.00 ETH	0	3	
0x379E65FB9f7A64498e9d53428B7d6060d7ad74D9	1000.00 ETH	0	4	

Connection Of MEW and Ganache

Step-10:

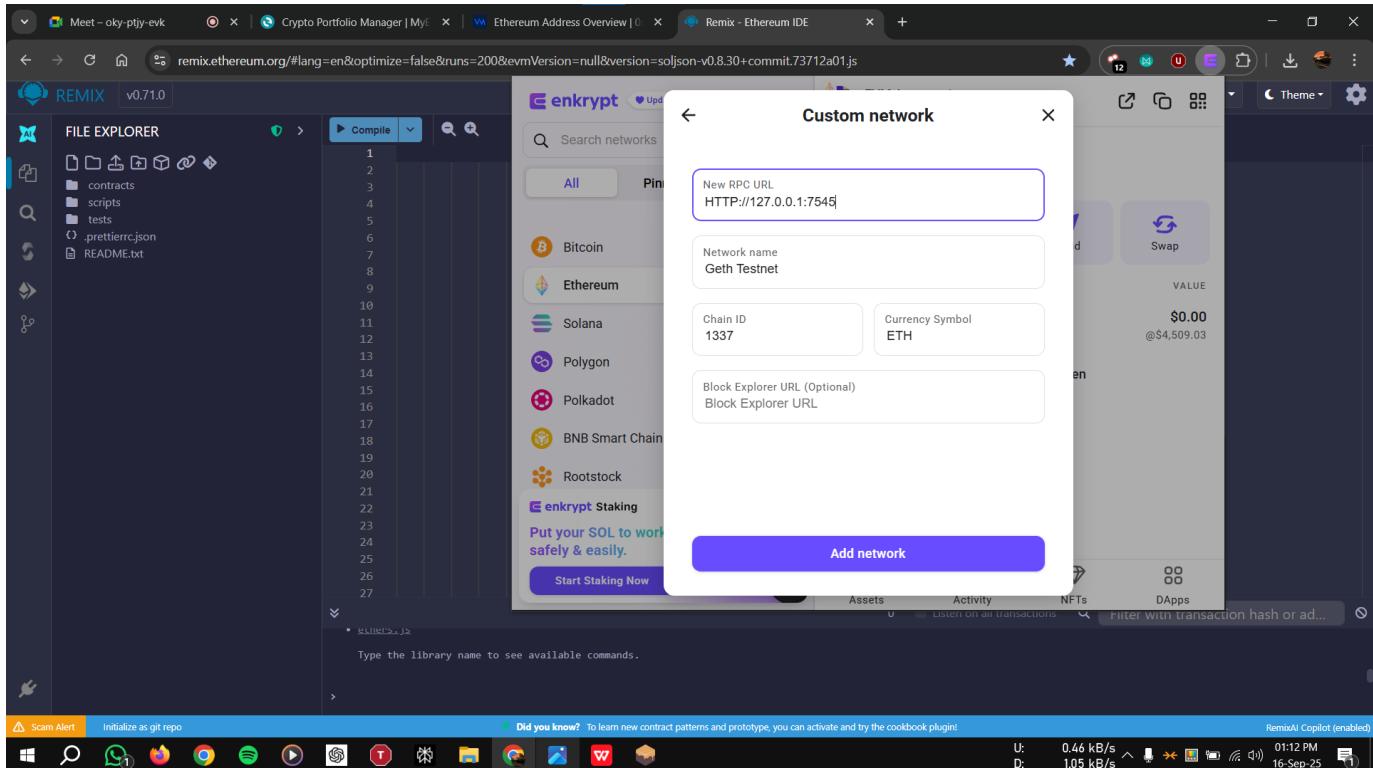
- First open the Encrypt in chrome and then click on other networks.



The screenshot shows the Enkrypt extension in the Chrome browser. The address bar shows the URL: remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.30+commit.73712a01.js. The browser window has several tabs open, including "Meet - oky-ptjy-evk", "Crypto Portfolio Manager | Myl", "Ethereum Address Overview | 0", and "Remix - Ethereum IDE". The Enkrypt extension is active, displaying a sidebar with network selection options: "All", "Lock Enkrypt", "Settings", and "Updates". The main panel shows an "EVM Account 1" with the address 0x52A7...63d3 and a balance of 0 ETH. It includes buttons for "Deposit", "Buy/Sell", "Send", and "Swap". Below this, it shows a token balance for Ethereum: 0 ETH, Last 24H: -2.70%, Value: \$0.00. There are also buttons for "Assets", "Activity", "NFTs", and "DApps". The bottom of the screen shows the Windows taskbar with various icons and network status information.

Step-11:

- Click on add Network & paste RPC url from Ganache app and you will get the network name Geth Testnet, After that click on add network to create a network.



Step-12: Now open the Remix IDE and create a file with the name prac4.sol.

Step-13: Now write insecure smart contract. As bellow given example.

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;
```

```
contract InsecureTransaction {
```

```
    event TransactionInitiated(address indexed sender, address indexed recipient, uint256 amount);
```

```
    function sendTransaction(address recipient, uint256 amount) public payable {
```

```
        // No balance check for sender
```

```
        // No validation for recipient address
```

```
        // Directly transfer amount
```

```
        payable(recipient).transfer(amount);
```

```
        emit TransactionInitiated(msg.sender, recipient, amount);
```

```
}
```

```
    // This function allows the contract to receive Ether
```

```
    receive() external payable {}
```

{}

Step-14:

- Now we need to compile this smart contract for that purpose click on the Solidity Compiler option and then set the compiler version according to the smart contract of yours.

Step-15:

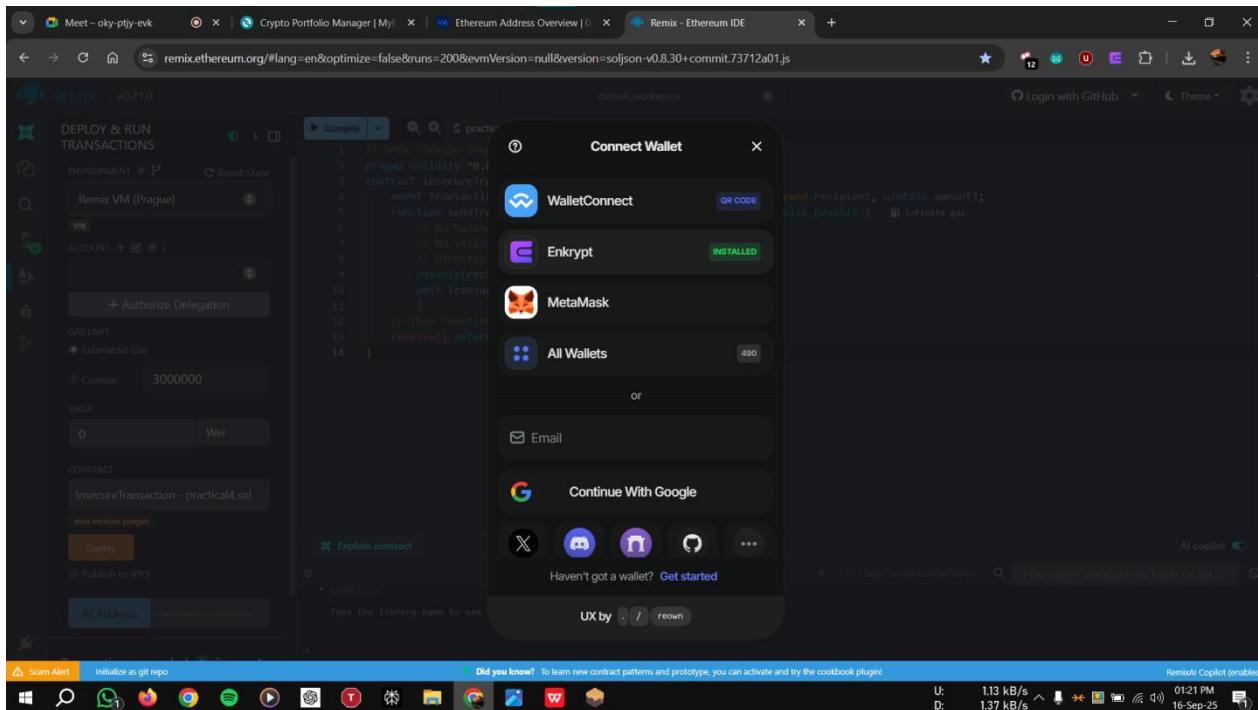
- After setting the compiler version click on compile.

Step-16:

- After Compiling now go to the deploy and in deploy, we need to connect the Enkrypt to the Remix IDE.

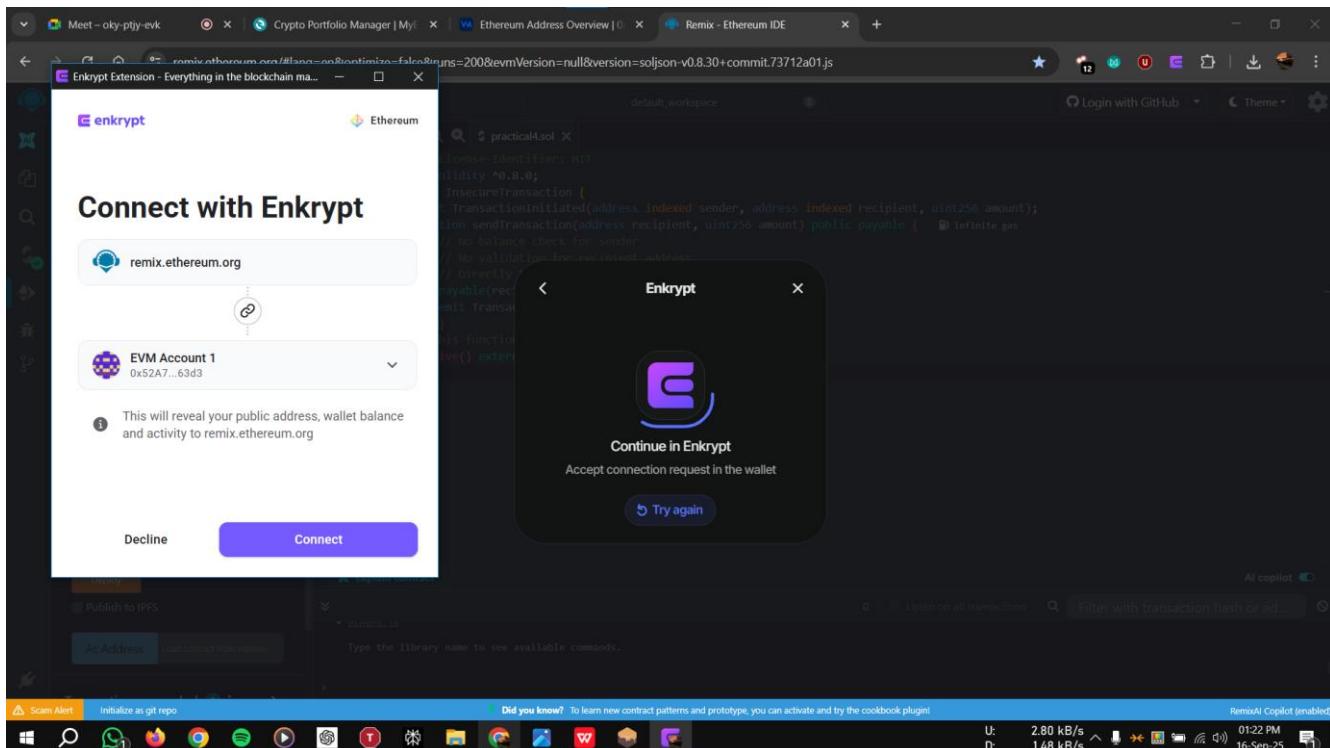
Step-17:

- To connect that Enkrypt with remix ide in deploy screen at the top we can able to see the Environment click on the dropdown and click on the Inject provider-Enkrypt.



Step-18:

- After clicking on that it will pop up one screen in that screen we need to click on the account which we had created and now click on the connect.

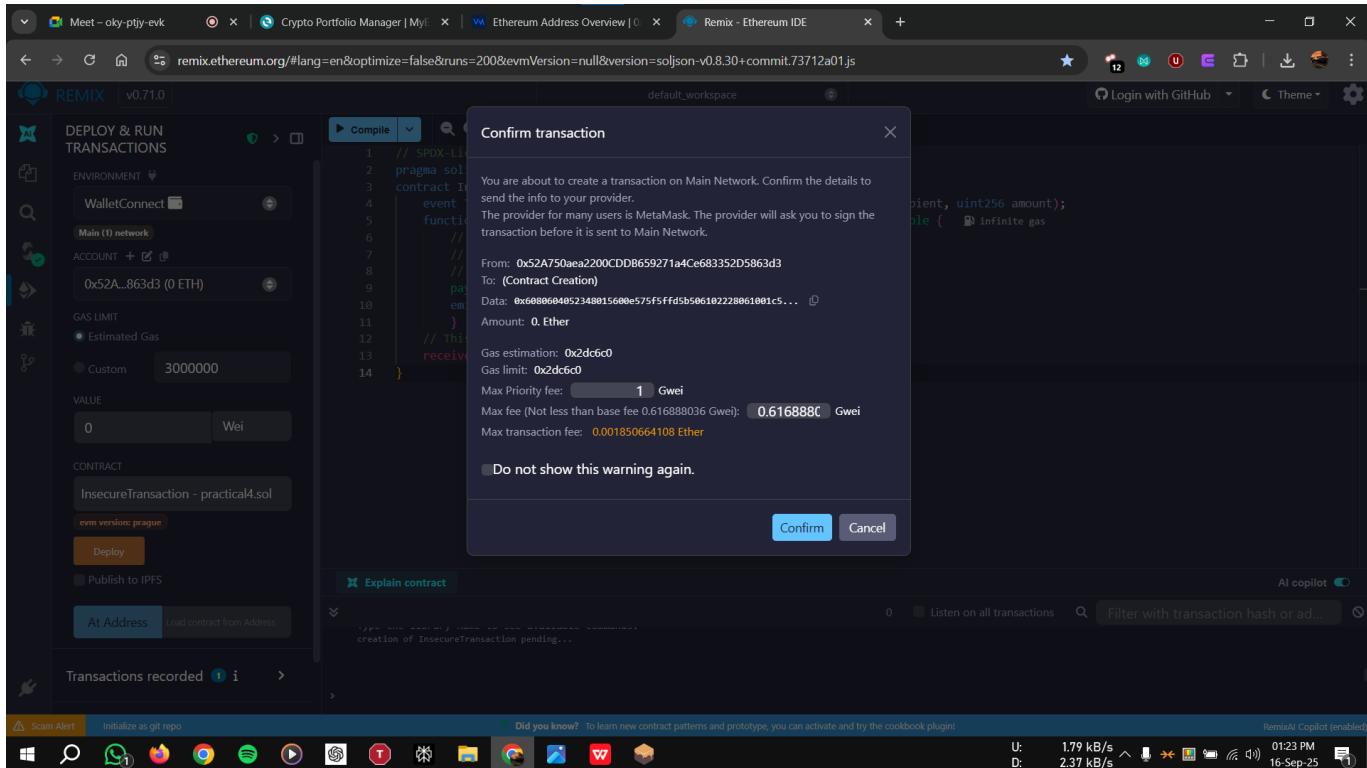


Step-19:

- After clicking that we can able to see the that account is added to it and we can able to see some ethers whatever present in that account.

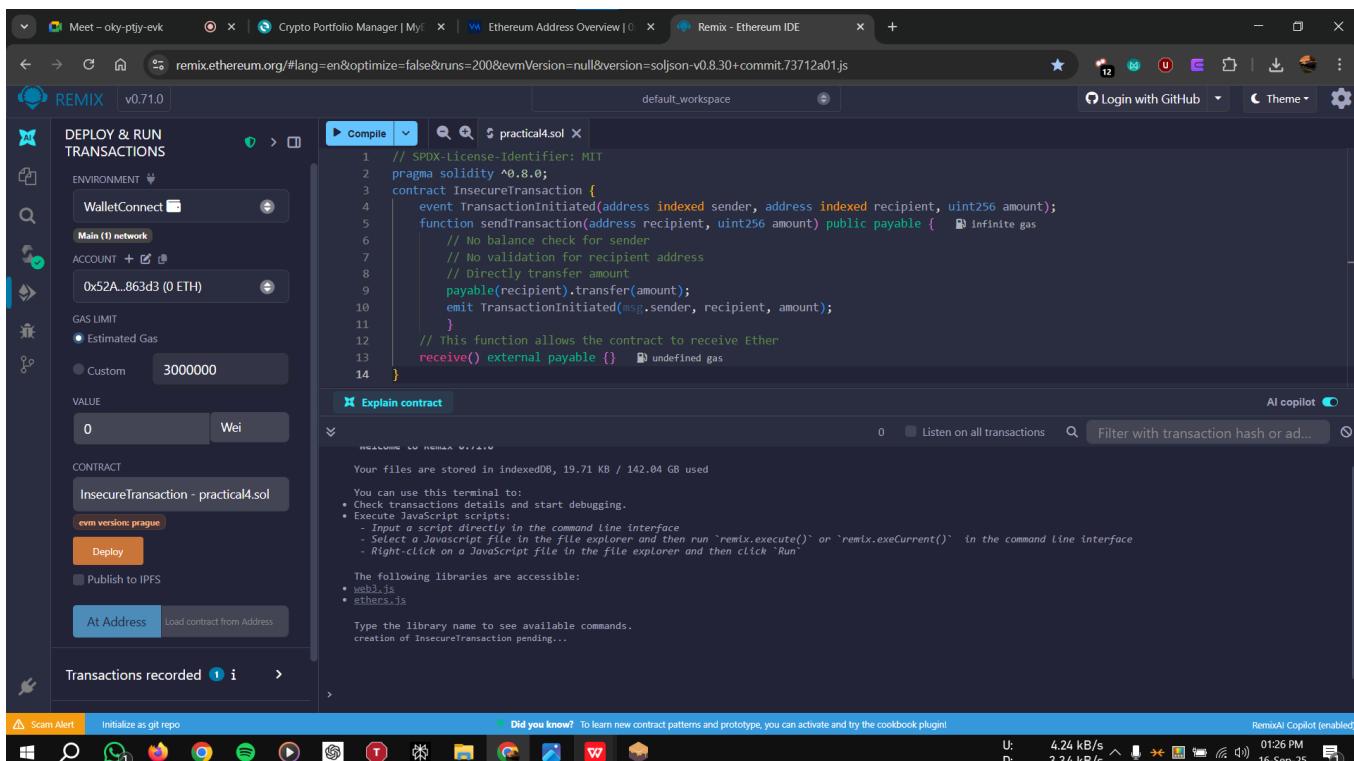
Step-20:

- Now click on the deploy button, the gas fee will generate and It will ask us to send that transaction or not.



Step-21:

- After clicking on the send the Smart contract is successfully deployed, now we can able to see the smart contract at the below of that deploy button.



Step-22:

- Now we need to give the account address to where we need to send the amount. Then will get an error because the it is an insecure smart contract.

Step-23:

- Now we need to update the smart contract and we need to compile and deploy that file again the secure smart contract is as follows.

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;
```

```
contract SecureTransaction {
```

```
    event TransactionInitiated(address indexed sender, address indexed recipient, uint256 amount);
```

```
    function sendTransaction(address recipient, uint256 amount) public {
```

```
        require(address(this).balance >= amount, "Insufficient contract balance");
```

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

```
        (bool success, ) = payable(recipient).call{value: amount}("");
```

```
        require(success, "Transaction failed");
```

```
        emit TransactionInitiated(msg.sender, recipient, amount);
```

```
}
```

```
    function getBalance() public view returns (uint256) { return address(this).balance;
```

```
}
```

```
    receive() external payable {}
```

```
}
```

Step-24:

- Now we need to give the account number near the empty space after the send_transaction.

```
contract SecureTransaction {
    event TransactionInitiated(address indexed sender, address indexed recipient, uint256 amount);

    function sendTransaction(address recipient, uint256 amount) public {
        require(address(this).balance >= amount, "Insufficient contract balance");
        require(recipient != address(0), "Invalid recipient address");

        (bool success, ) = payable(recipient).call{value: amount}("");
        require(success, "Transaction failed");

        emit TransactionInitiated(msp.sender, recipient, amount);
    }

    function getBalance() public view returns (uint256) {
        return address(this).balance;
    }

    receive() external payable {}
}
```

Conclusion:

Hence, we concluded the working of a secure smart contract as well as learnt how to setup a secure wallet with key management.

Practical No - 5

Aim: Introduction to Web 3.0 & Setting Up Basic DApp Environment

- Understand the evolution from Web 2.0 to Web 3.0.
- Explore decentralized applications (DApps).
- Set up a basic DApp environment using Ganache, Truffle, and MetaMask.

Requirements:

- Ganache (Ethereum Blockchain Simulator)
- Node.js
- Truffle Framework
- Visual Studio Code (or any text editor)
- MetaMask Wallet

Theory:

Understanding the Evolution from Web 2.0 to Web 3.0:

Web 2.0 vs Web 3.0

- **Web 2.0:**

Refers to the second generation of the World Wide Web.

Focuses on user-generated content, usability, and interoperability.

Emphasizes social networking sites, blogs, wikis, and video-sharing platforms where users interact and collaborate on content.

Example of Web 2.0:

Facebook: Allows users to create profiles, share content, and interact with others on a centralized platform controlled by Facebook.

- **Web 3.0:**

Known as the decentralized web.

Builds on Web 2.0 with a focus on decentralization, blockchain technology, and token-based economics.

Aims to give users more control over their data and digital identities.

Example of Web 3.0:

Ethereum-based social media platforms like Minds: Users own their data, earn tokens for contributions, and interact on a decentralized network.

Exploring Decentralized Applications (DApps)

What are DApps?

Applications that run on a blockchain or peer-to-peer network instead of centralized servers.

They are open-source, operate autonomously, and use smart contracts to enforce rules and interactions without a central authority.

Example of a DApp:

Uniswap: A decentralized exchange on the Ethereum blockchain that allows users to trade cryptocurrencies without a central intermediary.

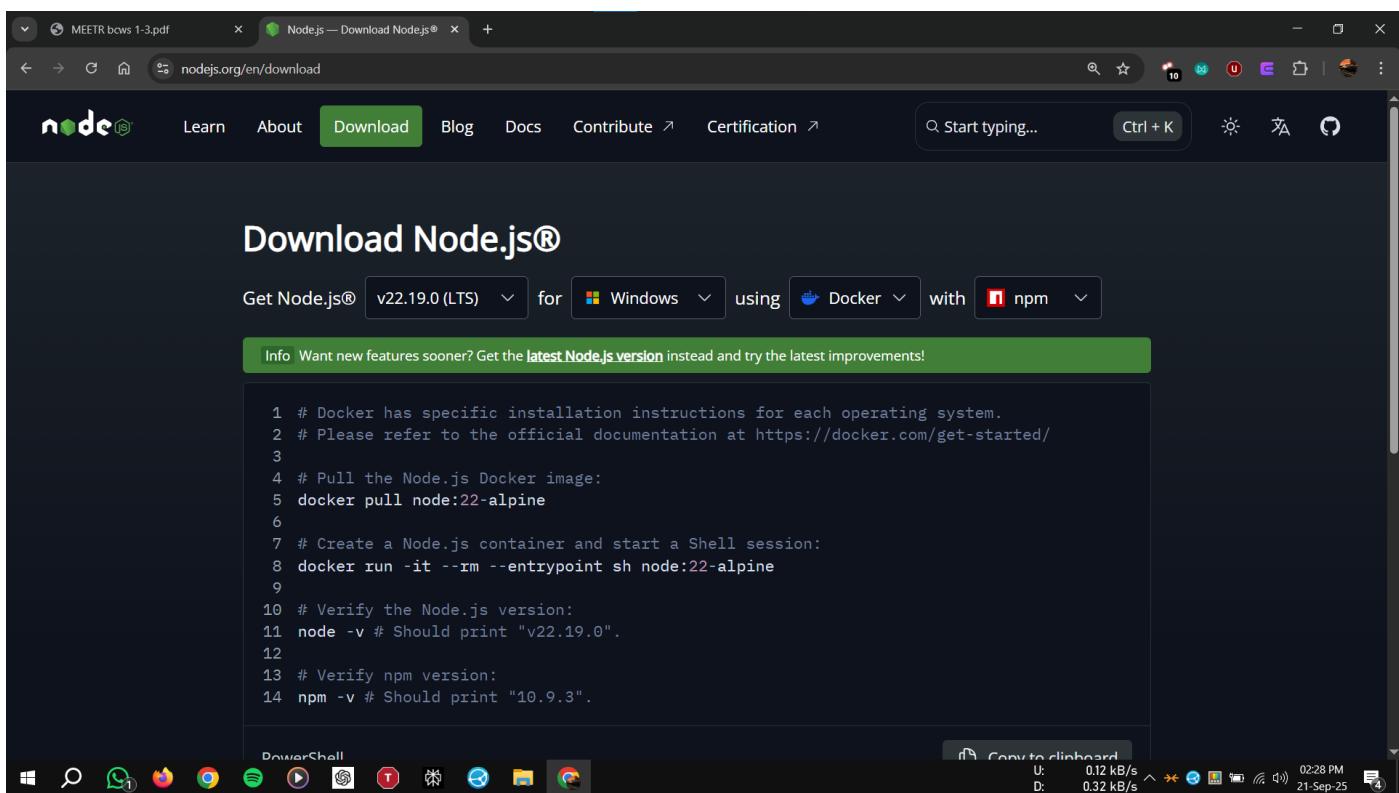
Procedure: Setting Up a Basic DApp Environment

Step 1: Install Node.js

Download Node.js from the official website: <https://nodejs.org>

Choose the LTS (Long Term Support) version for better stability.

Run the installer and complete the installation.



Step 2: Verify Node.js Installation

Open Command Prompt or Terminal.

Run the command:

`node --version`

It should display the installed Node.js version.

```
C:\Users\faltu>node --version  
v22.19.0
```

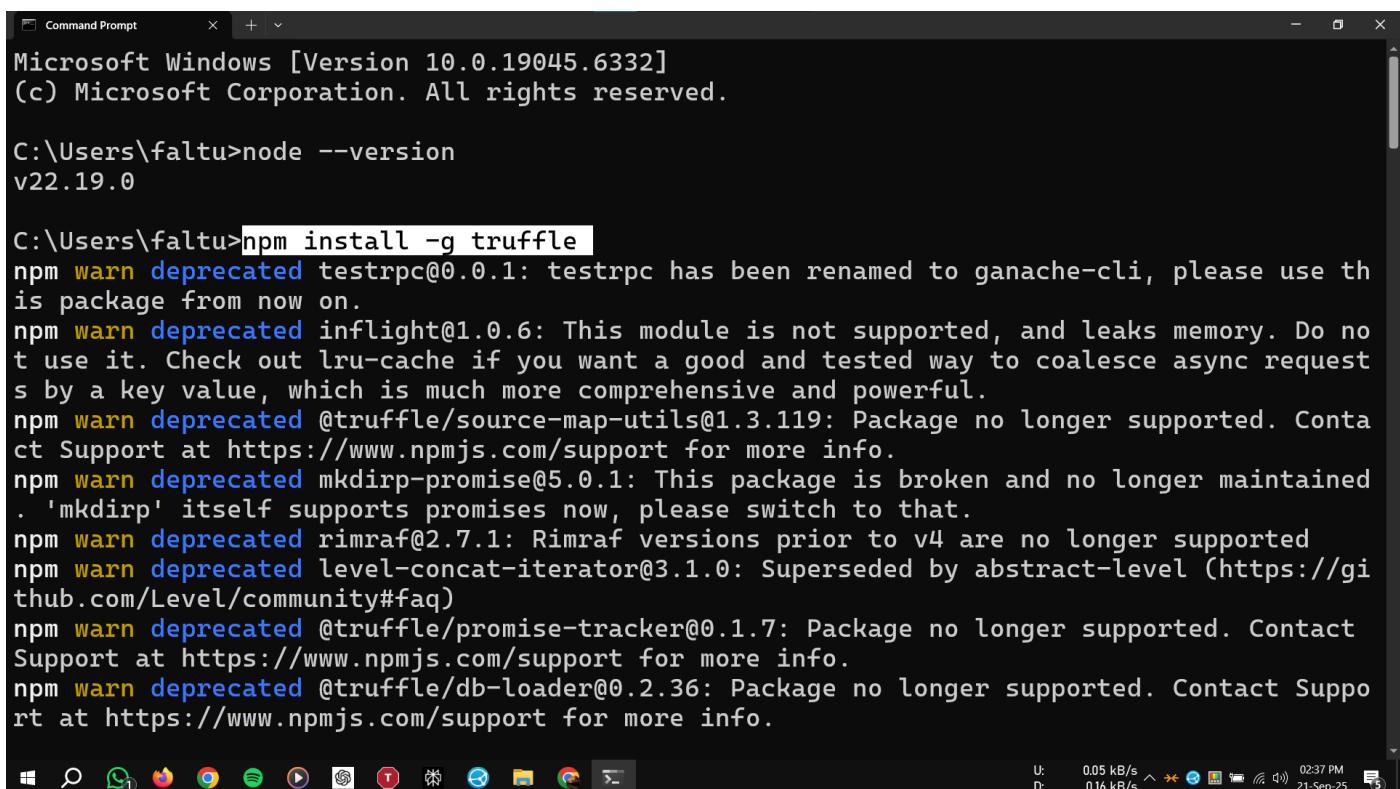
```
C:\Users\faltu>
```

Step 3: Install Truffle Framework

Run the following command in terminal:

```
npm install -g truffle
```

This installs Truffle globally.



```
Microsoft Windows [Version 10.0.19045.6332]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\faltu>node --version  
v22.19.0  
  
C:\Users\faltu>npm install -g truffle  
npm warn deprecated testrpc@0.0.1: testrpc has been renamed to ganache-cli, please use this package from now on.  
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.  
npm warn deprecated @truffle/source-map-utils@1.3.119: Package no longer supported. Contact Support at https://www.npmjs.com/support for more info.  
npm warn deprecated mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports promises now, please switch to that.  
npm warn deprecated rimraf@2.7.1: Rimraf versions prior to v4 are no longer supported  
npm warn deprecated level-concat-iterator@3.1.0: Superseded by abstract-level (https://github.com/Level/community#faq)  
npm warn deprecated @truffle/promise-tracker@0.1.7: Package no longer supported. Contact Support at https://www.npmjs.com/support for more info.  
npm warn deprecated @truffle/db-loader@0.2.36: Package no longer supported. Contact Support at https://www.npmjs.com/support for more info.  
  
U: 0.05 kB/s ^ D: 0.16 kB/s 02:37 PM 21-Sep-25
```

Step 4: Install and Open Ganache

Download Ganache from: <https://trufflesuite.com/ganache>

Install and open Ganache.

Click New Workspace.

Rename project to practical-5.

Go to Accounts & Keys.

Set number of accounts to 5.

Set balance for each account to 1000 ETH.

Click Start to run the local blockchain.

WORKSPACE

WORKSPACE NAME
practical-5

A friendly name for this workspace.

TRUFFLE PROJECTS

Link Truffle projects to this workspace by adding their truffle-config.js or truffle.js file to this workspace.

This will show useful contract and event data to better understand what's going on under the hood.

ADD PROJECT **REMOVE PROJECT**

ACCOUNTS & KEYS

ACCOUNT DEFAULT BALANCE
1000

The starting balance for accounts, in Ether.

TOTAL ACCOUNTS TO GENERATE
5

Total number of Accounts to create and pre-fund.

AUTOGENERATE HD MNEMONIC

Turn on to automatically generate a new mnemonic and account addresses on each run.

mnemonic: stumble nice wood jump news identify vanish park fish resource mask gain

Enter the Mnemonic you wish to use.

note: this mnemonic is not secure; don't use it on a public blockchain.

LOCK ACCOUNTS

If enabled, accounts will be locked on startup.



Step 5: Create Truffle Project

Open Visual Studio Code (VSCode).

Create a new folder (e.g., practical-5).

Open terminal inside VSCode at the project folder.

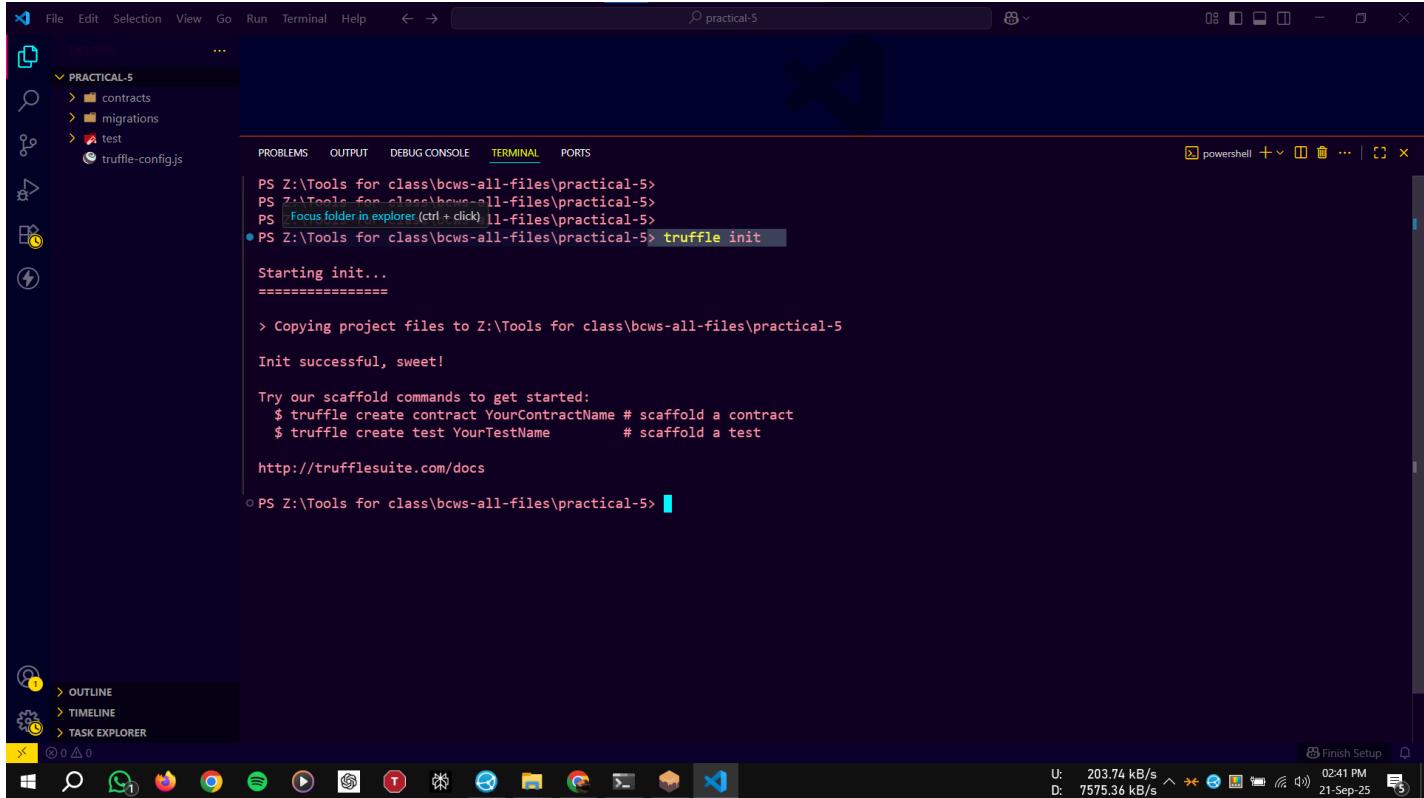
Initialize a Truffle project by running:

2203031260184

Page |

truffle init

This will create default folders and files.



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the output of the 'truffle init' command. The output includes:

```
PS Z:\Tools for class\bcws-all-files\practical-5>
PS Z:\Tools for class\bcws-all-files\practical-5>
PS Focus folder in explorer (ctrl + click) ll-files\practical-5>
● PS Z:\Tools for class\bcws-all-files\practical-5> truffle init

Starting init...
=====
> Copying project files to Z:\Tools for class\bcws-all-files\practical-5

Init successful, sweet!

Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName           # scaffold a test

http://trufflesuite.com/docs

○ PS Z:\Tools for class\bcws-all-files\practical-5>
```

The terminal window has a dark theme. The status bar at the bottom shows network activity and system time.

Step 6: Configure Truffle to Connect with Ganache

Open the `truffle-config.js` file.

Locate the `networks` section.

Uncomment and add this configuration:

```
networks: {
  development: {
    host: "127.0.0.1",
    port: 7545,          // Ganache default port
    network_id: "*",    // Match any network id
  },
}
```

Save the file.

```

49 module.exports = {
50   /*
51   networks: {
52     // Useful for testing. The `development` name is special - truffle uses it by default
53     // if it's defined here and no other network is specified at the command line.
54     // You should run a client (Like ganache, geth, or parity) in a separate terminal
55     // tab if you use this network and you must also set the `host`, `port` and `network_id`
56     // options below to some value.
57     //
58     development: {
59       host: "127.0.0.1", // Localhost (default: none)
60       port: 7545, // Standard Ethereum port (default: none)
61       network_id: "*", // Any network (default: none)
62     },
63     //
64     // An additional network, but with some advanced options...
65     // advanced: {
66     //   port: 8777, // Custom port
67     //   network_id: 1342, // Custom network
68     //   gas: 8500000,
69     //   gasPrice: 2000000000, // 20 gwei (in wei) (default: 100 gwei)
70     //   from: <address>, // Account to send transactions from (default: accounts[0])
71   },
72 }
73 =====
74 > Copying project files to Z:\Tools for class\bcws-all-files\practical-5
75 Init successful, sweet!
76 Try our scaffold commands to get started:

```

Step 7: Create Smart Contract

Inside the contracts folder, create a new file named MyContract.sol.

Write a simple contract, for example:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private storedNumber;

    //function to store a number
    function store(uint256 _number) public{
        storedNumber = _number;
    }

    //function to retrieve the stored number
    function retrieve() public view returns (uint256) {
        return storedNumber;
    }
}

```

Save the contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleStorage {
    uint256 private storedNumber;
    //function to store a number
    function store(uint256 _number) public{
        storedNumber = _number;
    }

    //function to retrieve the stored number
    function retrieve() public view returns (uint256) {
        return storedNumber;
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
> Copying project files to Z:\Tools for class\bcws-all-files\practical-5
Init successful, sweet!
Try our scaffold commands to get started:
```

Ln 15, Col 9 Spaces: 4 UTF-8 CRLF ⚙ Solidity Finish Setup 02:50 PM 21-Sep-25

Step 8: Write Migration Script

Open the migrations folder.

Create a new file named `2_deploy_smartcontract.js`.

Add the following code:

```
const MyContract = artifacts.require("MyContract");
module.exports = function (deployer) {
    deployer.deploy(MyContract, "Hello, Blockchain!");
};
```

Save the file.

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a project named "practical-5" with subfolders "contracts" (containing ".gitkeep" and "mycontract.sol"), "migrations" (containing ".gitkeep" and "2_deploy_smartcontract.js"), and "test" (containing "truffle-config.js").
- Code Editor:** Displays the file "2_deploy_smartcontract.js" with the following code:


```
1 const MyContract = artifacts.require("MyContract");
2 module.exports = function (deployer) {
3   deployer.deploy(MyContract, "Hello, Blockchain!");
4 }
```
- Terminal:** Shows the output of running Truffle commands:


```
> Copying project files to Z:\Tools for class\bcws-all-files\practical-5
Init successful, sweet!
Try our scaffold commands to get started:
```
- Bottom Status Bar:** Shows file paths (Ln 4, Col 3), file statistics (U: 0.12 kB/s, D: 0.76 kB/s), and a timestamp (02:52 PM, 21-Sep-25).

Step 9: Compile and Deploy Contract

In terminal, run:

truffle compile

This compiles your smart contracts.

Then run:

truffle migrate --network development

This deploys the contract to Ganache blockchain.

The terminal window shows the following command and its output:

```
http://trufflesuite.com/docs
PS Z:\Tools for class\bcws-all-files\practical-5> truffle compile
● Compiling your contracts...
=====
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling .\contracts\mycontract.sol
> Artifacts written to Z:\Tools for class\bcws-all-files\practical-5\build\contracts
> Compiled successfully using:
  - solc: 0.8.21+commit.d9974bed.Emscripten clang
○ PS Z:\Tools for class\bcws-all-files\practical-5>
```

```

PS Z:\Tools for class\bcws-all-files\practical-5> truffle migrate --network development
This version of μWS is not compatible with your Node.js build:

Error: Cannot find module '../binaries/uws_win32_x64_127.node'
Require stack:
- C:\Users\faltu\AppData\Roaming\npm\node_modules\truffle\node_modules\ganache\node_modules\@trufflesuite\uws-js-unofficial\src\uws.js
- Open file in editor (ctrl + click)\Roaming\npm\node_modules\truffle\node_modules\ganache\dist\node\core.js
- C:\Users\faltu\AppData\Roaming\npm\node_modules\truffle\build\migrate.bundled.js
- C:\Users\faltu\AppData\Roaming\npm\node_modules\truffle\node_modules\original-require\index.js
- C:\Users\faltu\AppData\Roaming\npm\node_modules\truffle\build\cli.bundled.js
Falling back to a NodeJS implementation; performance may be degraded.

Compiling your contracts...
=====
> Compiling .\contracts\mycontract.sol
> Artifacts written to Z:\Tools for class\bcws-all-files\practical-5\build\contracts
> Compiled successfully using:

```

Step 10: Verify Deployment on Ganache

Open Ganache GUI.

NAME	ADDRESS	TX COUNT
SimpleStorage	Not Deployed	0



Conclusion:

We learned about Web 3.0 and how it differs from Web 2.0.

Set up a local blockchain environment using Ganache.

Created, compiled, and deployed a smart contract using Truffle.

Connected your DApp environment to Ganache to simulate blockchain transactions.

Practical No - 6

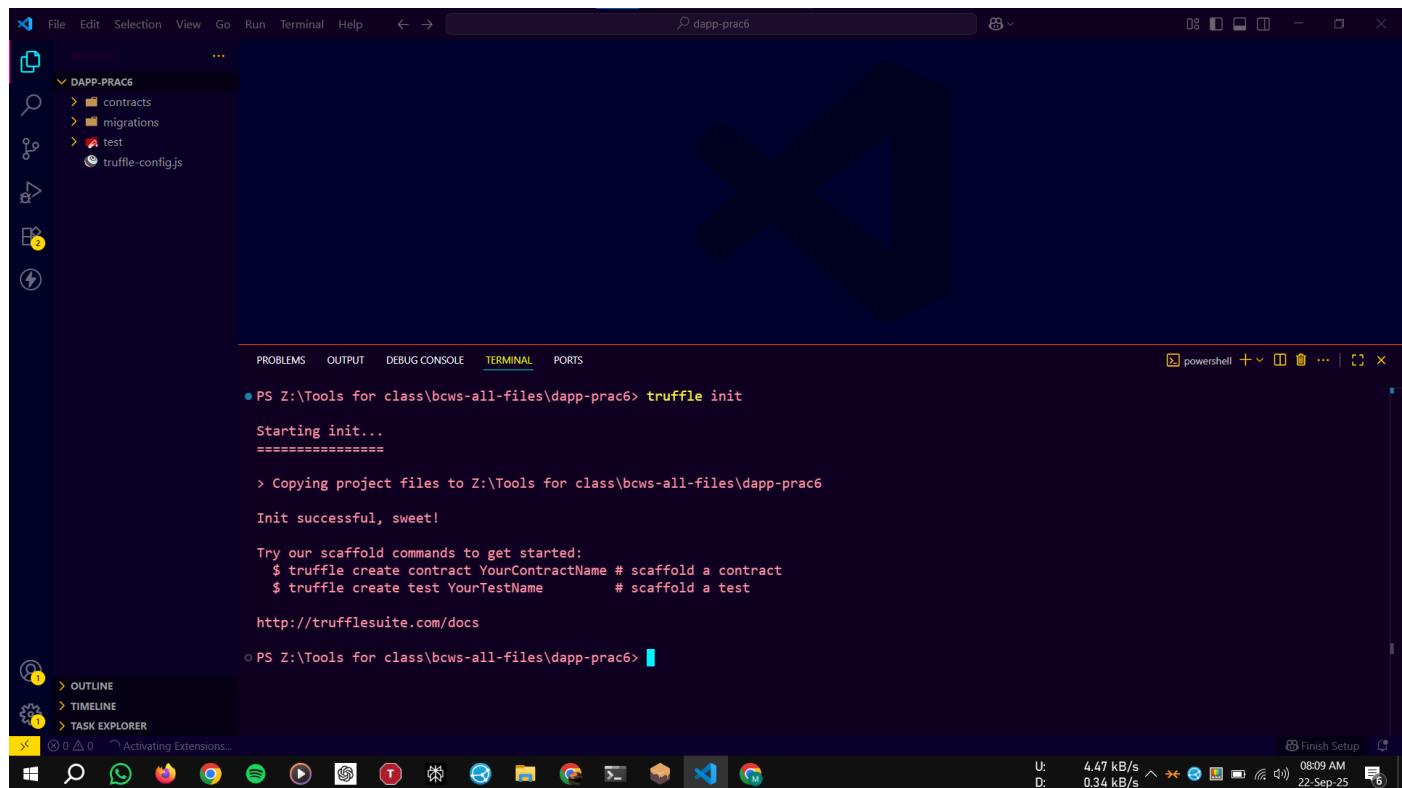
Aim: Developing and Deploying a Simple DApp.

1. Create a decentralized application (DApp) using a blockchain platform.
2. Deploy the DApp on a test network.
3. Interact with the DApp through a web interface.

Steps to create simple Dapp

Step-1

Open new folder in vscode and first initialize truffle project into this directory by giving the following command "*truffle init*"



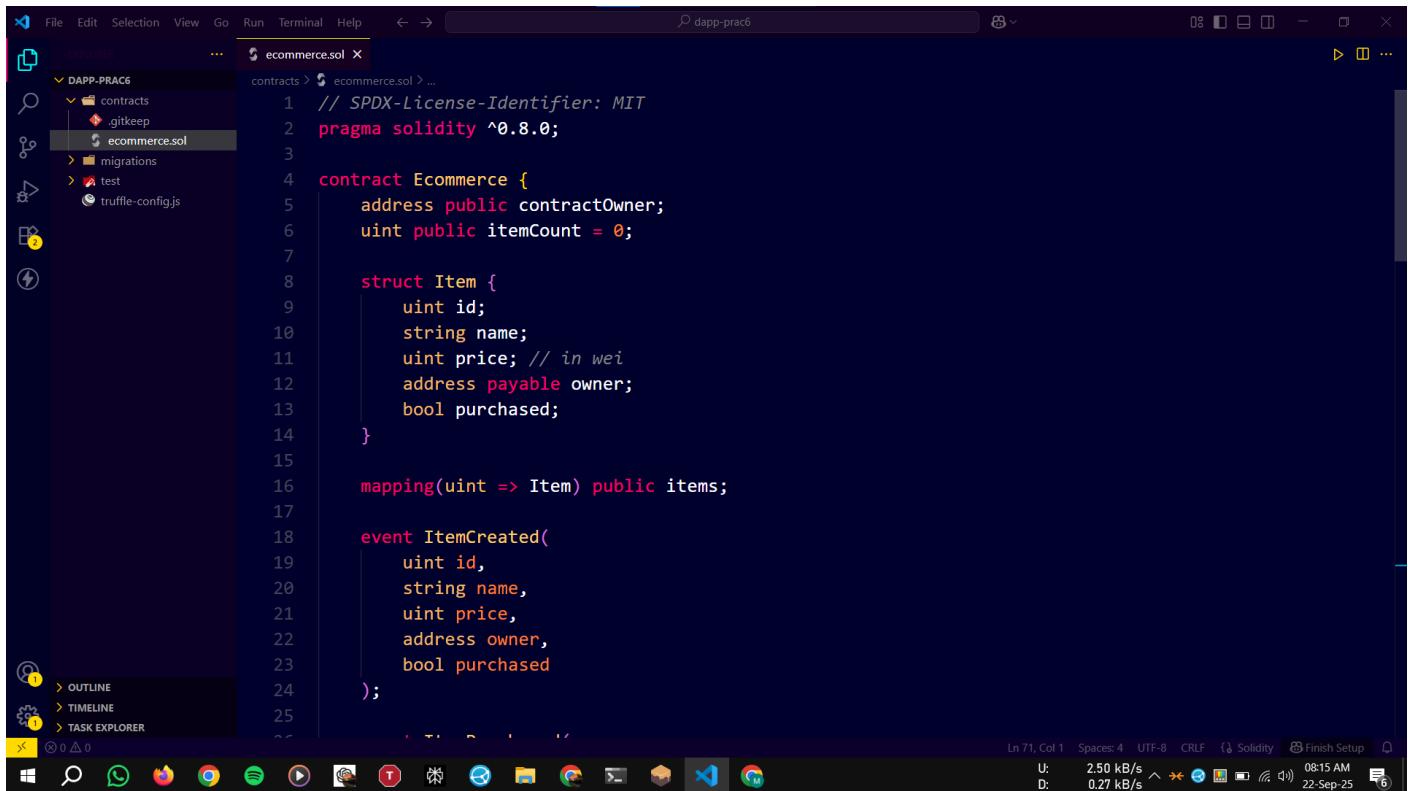
The screenshot shows a Windows desktop environment with VS Code open. The terminal tab is active, displaying the command "truffle init" being run in a PowerShell window. The output shows the initialization process, including copying project files and providing scaffold commands. The status bar at the bottom right shows network activity and system information.

```
PS Z:\Tools for class\bcws-all-files\dapp-prac6> truffle init
Starting init...
=====
> Copying project files to Z:\Tools for class\bcws-all-files\dapp-prac6
Init successful, sweet!

Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName           # scaffold a test
http://trufflesuite.com/docs
PS Z:\Tools for class\bcws-all-files\dapp-prac6>
```

Step-2

Now create a file in the contracts folder and name it as "*ecommerce.sol*" and write the smart contract into it.



```

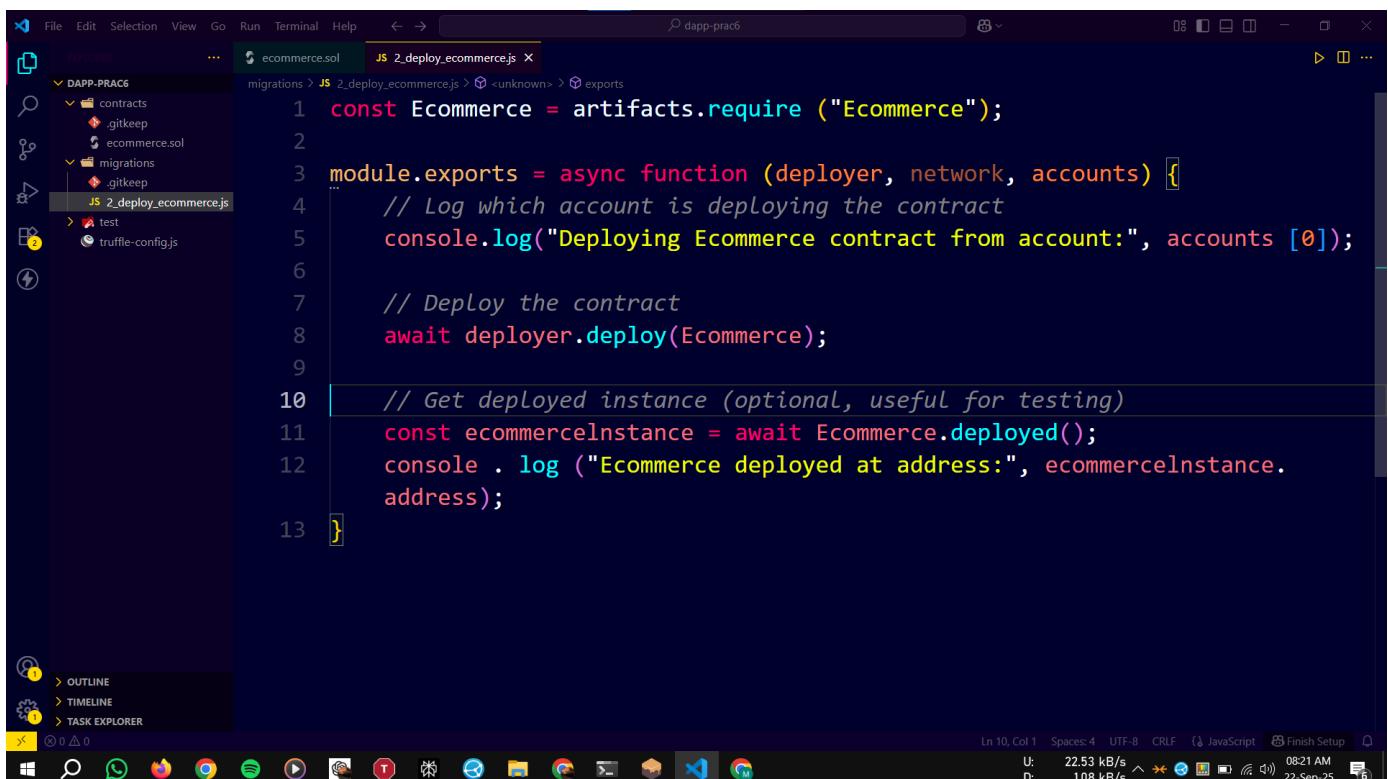
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Ecommerce {
5     address public contractOwner;
6     uint public itemCount = 0;
7
8     struct Item {
9         uint id;
10        string name;
11        uint price; // in wei
12        address payable owner;
13        bool purchased;
14    }
15
16    mapping(uint => Item) public items;
17
18    event ItemCreated(
19        uint id,
20        string name,
21        uint price,
22        address owner,
23        bool purchased
24    );
25

```

VS Code status bar: Ln 71, Col 1 Spaces: 4 UTF-8 CRLF (Solidity) Finish Setup
U: 2.50 kB/s D: 0.27 kB/s 08:15 AM 22-Sep-25

Step-3

Now open migration folder and create a new file and name it “`2_deploy_ecommerce.js`” and write the following code into it.



```

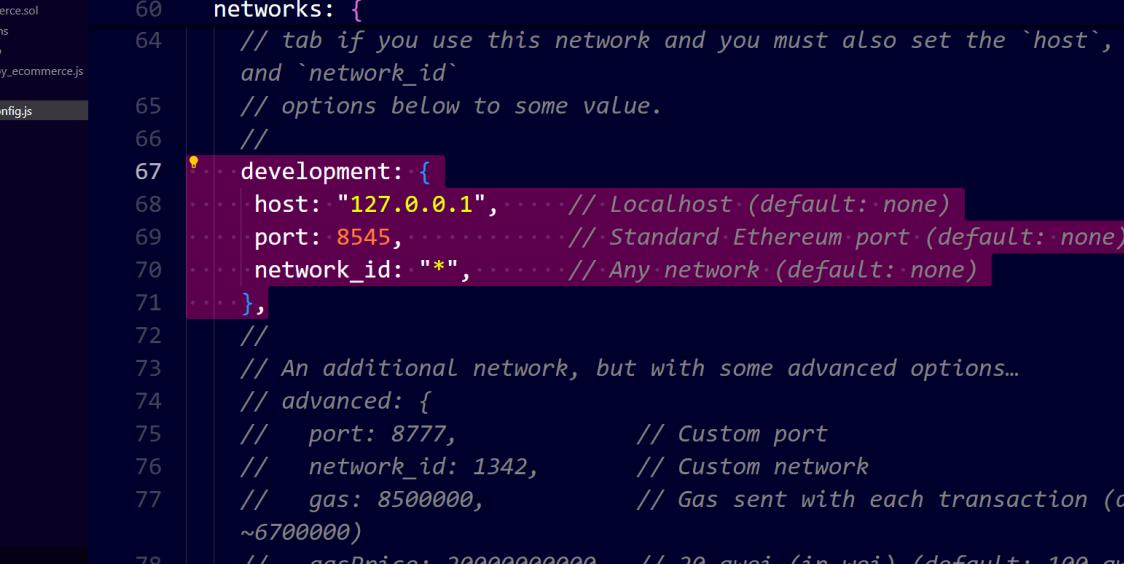
1 const Ecommerce = artifacts.require ("Ecommerce");
2
3 module.exports = async function (deployer, network, accounts) {
4     // Log which account is deploying the contract
5     console.log("Deploying Ecommerce contract from account:", accounts [0]);
6
7     // Deploy the contract
8     await deployer.deploy(Ecommerce);
9
10    // Get deployed instance (optional, useful for testing)
11    const ecommerceInstance = await Ecommerce.deployed();
12    console . log ("Ecommerce deployed at address:", ecommerceInstance.
13    address);
14

```

VS Code status bar: Ln 10, Col 1 Spaces: 4 UTF-8 CRLF (JavaScript) Finish Setup
U: 22.53 kB/s D: 1.08 kB/s 08:21 AM 22-Sep-25

Step-4

Now open *truffle-config.js* file and uncomment the following lines in that code as we did in the practical 5.



```
File Edit Selection View Go Run Terminal Help ← → 🔍 dapp-prac6 🌐 DAPP-PRAC6 contracts .gitkeep ecommerce.sol migrations .gitkeep JS 2_deploy_ecomerce.js test truffle-config.js 49 module.exports = { 50   networks: { 51     // tab if you use this network and you must also set the `host`, `port` 52     // and `network_id` 53     // options below to some value. 54     // 55     development: { 56       host: "127.0.0.1", // Localhost (default: none) 57       port: 8545, // Standard Ethereum port (default: none) 58       network_id: "*", // Any network (default: none) 59     }, 60     // 61     // An additional network, but with some advanced options... 62     // advanced: { 63       // port: 8777, // Custom port 64       // network_id: 1342, // Custom network 65       // gas: 8500000, // Gas sent with each transaction (default: 66       // ~6700000) 67       // gasPrice: 2000000000, // 20 gwei (in wei) (default: 100 gwei) 68       // from: <address>, // Account to send transactions from 69     } 70   } 71 } 72 // 73 // An additional network, but with some advanced options... 74 // advanced: { 75   // port: 8777, // Custom port 76   // network_id: 1342, // Custom network 77   // gas: 8500000, // Gas sent with each transaction (default: 78   // ~6700000) 79   // gasPrice: 2000000000, // 20 gwei (in wei) (default: 100 gwei) 80   // from: <address>, // Account to send transactions from
```

Step-5

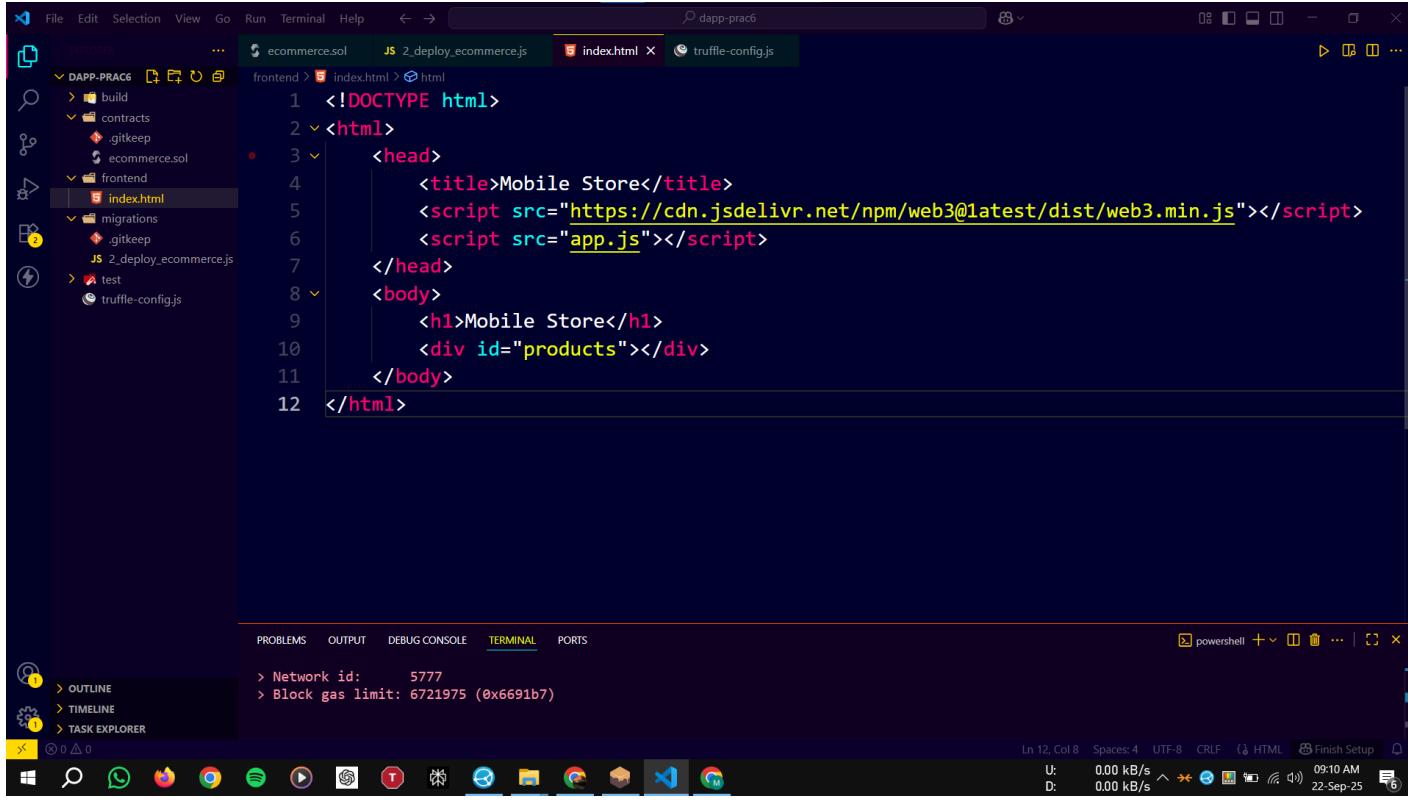
Now open terminal and give the following command “*truffle compile*” and “*truffle migrate*” as follows.

```
=====
Deploying Ecommerce contract from account: 0x666fde2A68111715Cbf016be49752f2a7430E6Ad

Replacing 'Ecommerce'
-----
> transaction hash: 0x33816ccdbc1096a1d0f0aeb3b76fec0a2cc330ead4e369cf0782823db6ddd31
> Blocks: 0 Seconds: 0
> contract address: 0xf475da29974c79D8e2999Bb8eDa63bbB4033144b
> block number: 2
> block timestamp: 1757336989
> account: 0x666fde2A68111715Cbf016be49752f2a7430E6Ad
> balance: 99.993579745666715329
> gas used: 962277 (0xeaee5)
> gas price: 3.296939923 gwei
> value sent: 0 ETH
> total cost: 0.003172569458284671 ETH
```

Step-6

Now lets create the frontend for the Dapp • now lets create frontend folder an dnow create index.html file in I as follows.



```

File Edit Selection View Go Run Terminal Help < → ecommerce.sol JS 2_deploy_ecomerce.js index.html truffle-config.js
frontend > index.html > index.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Mobile Store</title>
5     <script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
6     <script src="app.js"></script>
7   </head>
8   <body>
9     <h1>Mobile Store</h1>
10    <div id="products"></div>
11  </body>
12 </html>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

> Network id: 5777
> Block gas limit: 6721975 (0x6691b7)

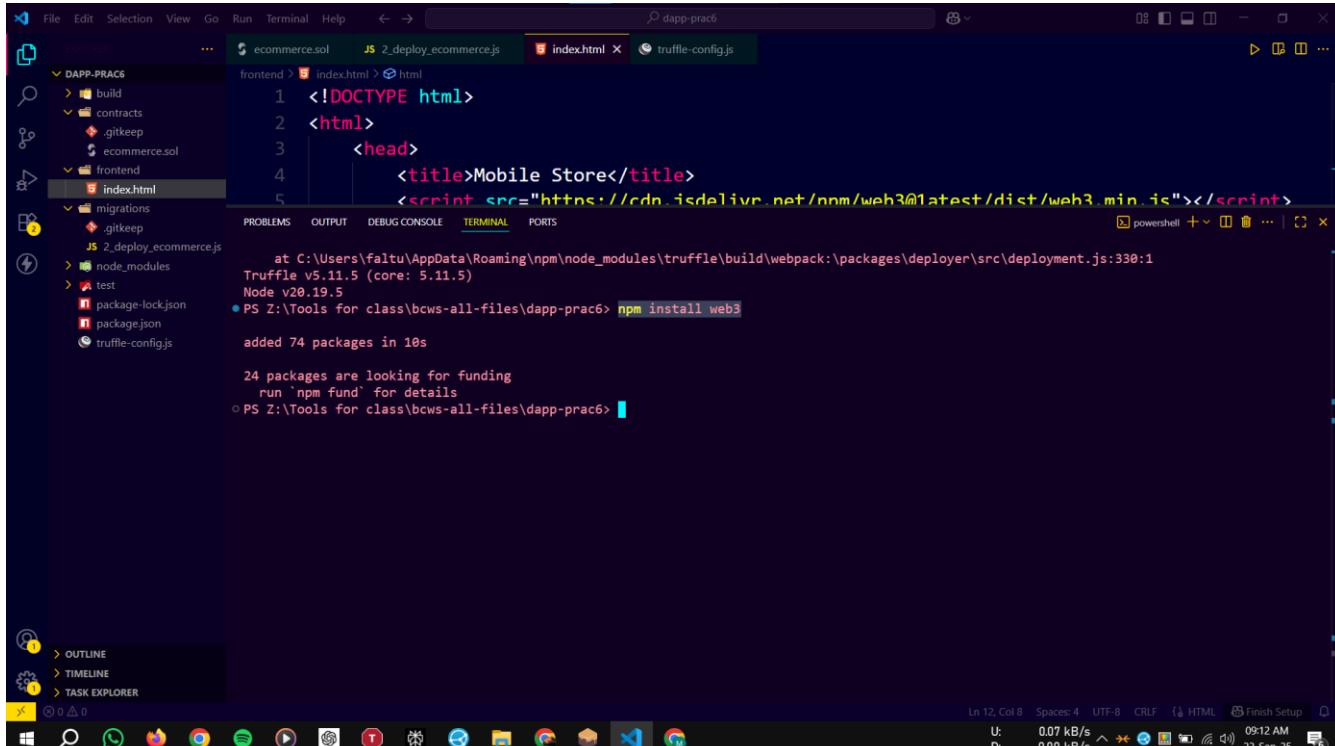
Ln 12, Col 8 Spaces: 4 UTF-8 CRLF ⚡ HTML ⚡ Finish Setup

U: 0.00 kB/s D: 0.00 kB/s 09:10 AM 22-Sep-25

Step-7

Now create styles.css file.

lets download web3 by using command "*npm install web3*" as follows



```

File Edit Selection View Go Run Terminal Help < → ecommerce.sol JS 2_deploy_ecomerce.js index.html truffle-config.js
frontend > index.html > index.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Mobile Store</title>
5     <script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
6
7   at C:\Users\faltu\AppData\Roaming\npm\node_modules\truffle\build\webpack\packages\deployer\src\deployment.js:330:1
8 Truffle v5.11.5 (core: 5.11.5)
9 Node v20.19.5
10 ● PS Z:\Tools for class\bcws-all-files\dapp-prac6> npm install web3
11
12 added 74 packages in 10s
13
14 24 packages are looking for funding
15   run 'npm fund' for details
16
17 ○ PS Z:\Tools for class\bcws-all-files\dapp-prac6>

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

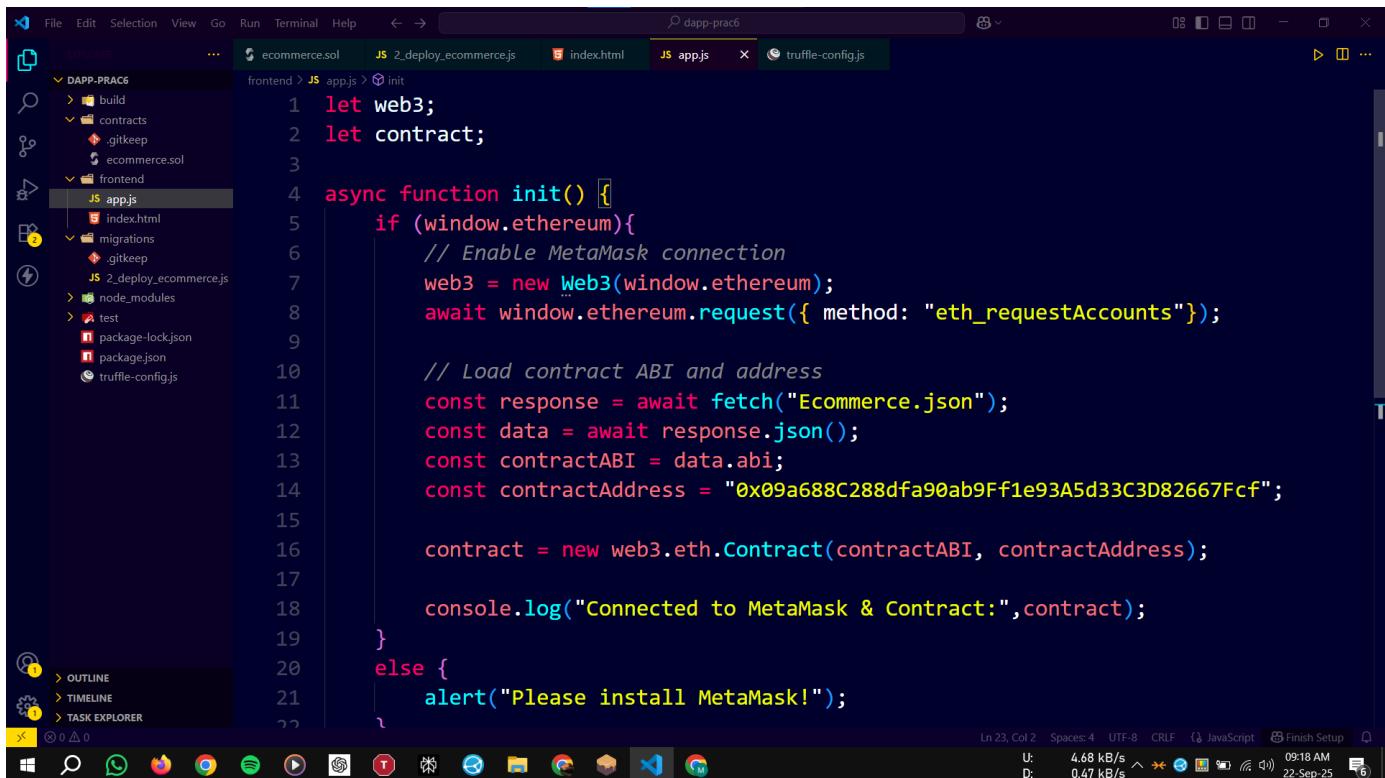
Ln 12, Col 8 Spaces: 4 UTF-8 CRLF ⚡ HTML ⚡ Finish Setup

U: 0.07 kB/s D: 0.00 kB/s 09:12 AM 22-Sep-25

Step-9

Now create app.js file.

In this file give the contract address which we had got in terminal we we migrated the truffle.



```

File Edit Selection View Go Run Terminal Help < > dapp-prac6
DAPP-PRAC6 ecommerce.sol JS 2_deploy_eCommerce.js index.html JS app.js truffle-config.js
frontend > JS app.js > init
1 let web3;
2 let contract;
3
4 async function init() {
5   if (window.ethereum){
6     // Enable MetaMask connection
7     web3 = new Web3(window.ethereum);
8     await window.ethereum.request({ method: "eth_requestAccounts"});
9
10    // Load contract ABI and address
11    const response = await fetch("Ecommerce.json");
12    const data = await response.json();
13    const contractABI = data.abi;
14    const contractAddress = "0x09a688C288dfa90ab9FF1e93A5d33C3D82667FcF";
15
16    contract = new web3.eth.Contract(contractABI, contractAddress);
17
18    console.log("Connected to MetaMask & Contract:",contract);
19  }
20  else {
21    alert("Please install MetaMask!");
22  }
}

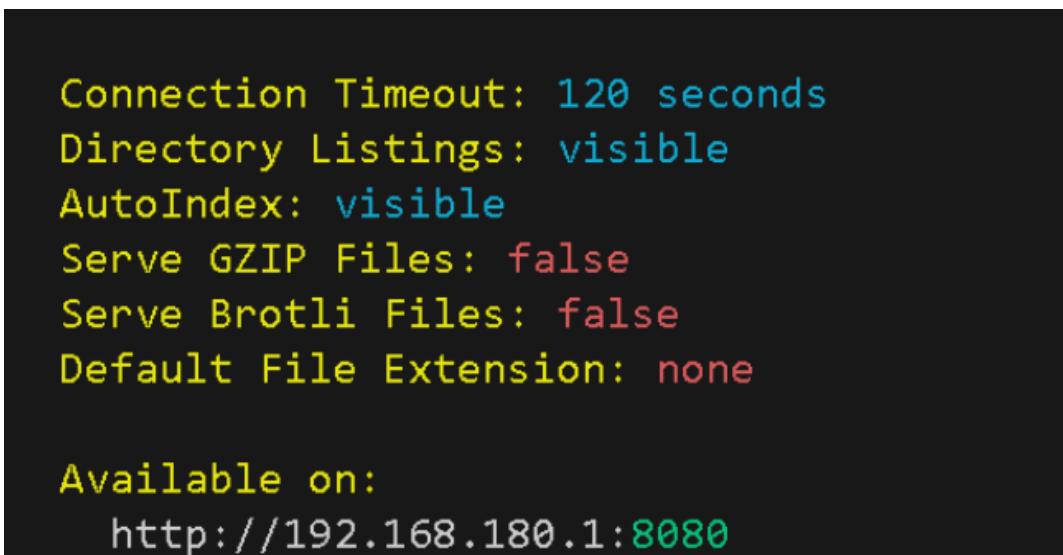
```

Ln 23, Col 2 Spaces: 4 UTF-8 CRLF JavaScript Finish Setup

U: 4.68 kB/s D: 0.47 kB/s 09:18 AM 22-Sep-25

Step-10

Now open terminal and give command npx http-server to run the Dapp in browser



```

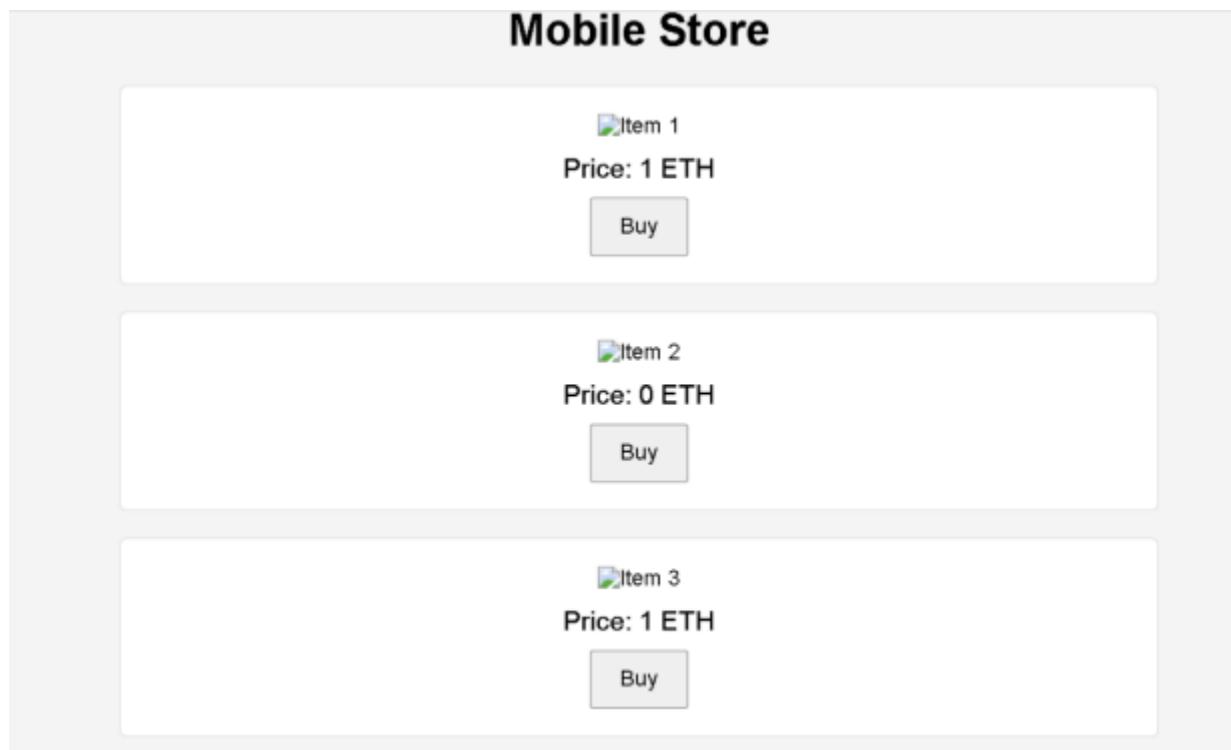
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
http://192.168.180.1:8080

```

Step-11

Now click on the give links and open it now ouor webpage will be open



Conclusion:

In this practical we learn and perform how to Create a decentralized application (DApp) using a blockchain platform. And Deploy the DApp on a test network. and Interact with the DApp through a web interface.

Practical - 7

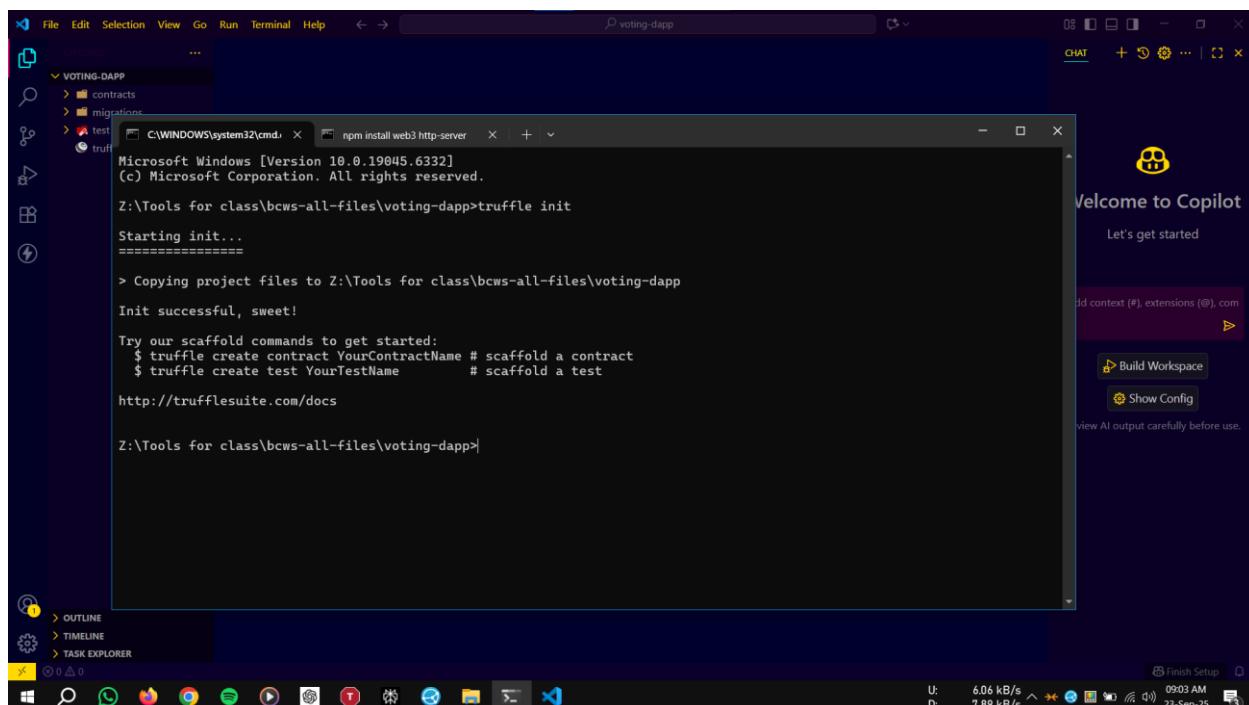
- Aim:**
1. Integrating blockchain functionality into web applications
 2. Implementing secure communication between web and blockchain components

Step1: Create a New Truffle Project

```
mkdir voting-dapp
```

```
cd voting-dapp
```

```
truffle init
```

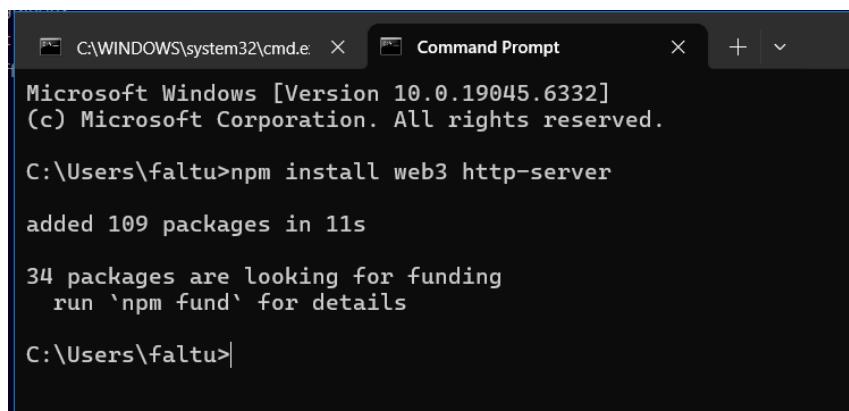


```
Z:\Tools for class\bcws-all-files\voting-dapp>truffle init
Starting init...
=====
> Copying project files to Z:\Tools for class\bcws-all-files\voting-dapp
Init successful, sweet!
Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName      # scaffold a test
http://trufflesuite.com/docs
Z:\Tools for class\bcws-all-files\voting-dapp>
```

Step2: Install Dependencies

Install `web3.js` and `http-server` for blockchain communication and serving the frontend

```
npm install web3 http-server
```



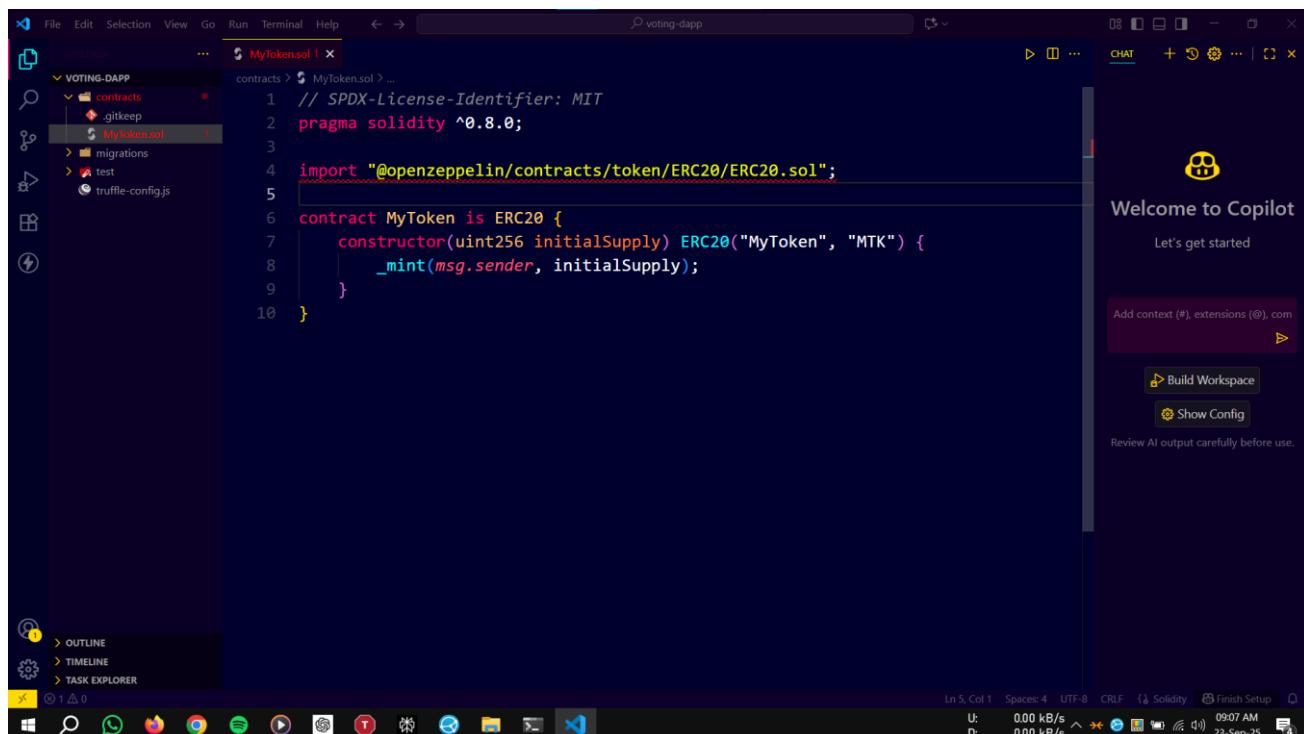
```
C:\WINDOWS\system32\cmd.e  X  Command Prompt  X  +
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

C:\Users\faltu>npm install web3 http-server
added 109 packages in 11s
34 packages are looking for funding
  run 'npm fund' for details
C:\Users\faltu>
```

Step3: Create the Smart Contracts

Create a new file `MyToken.sol` in the `contracts` directory:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
contract MyToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("MyToken", "MTK") {
        _mint(msg.sender, initialSupply);
    }
}
```



Create a new file `Voting.sol` in the `contracts` directory:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
contract Voting {
    IERC20 public token;
    mapping(address => uint256) public votes;
    address[] public candidates;
    constructor(IERC20 _token, address[] memory _candidates) {
        token = _token;
        candidates = _candidates;
    }
}
```

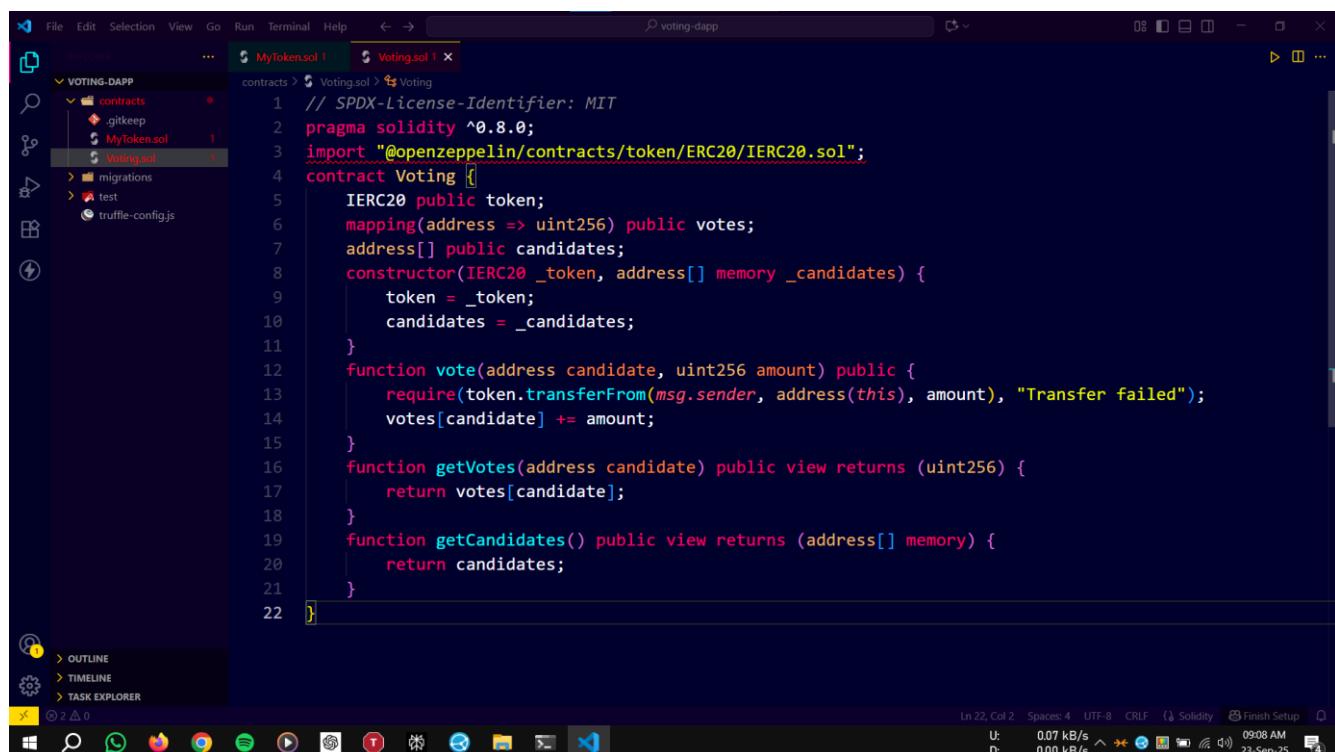
```
}

function vote(address candidate, uint256 amount) public {
    require(token.transferFrom(msg.sender, address(this), amount), "Transfer failed");
    votes[candidate] += amount;
}

function getVotes(address candidate) public view returns (uint256) {
    return votes[candidate];
}

function getCandidates() public view returns (address[] memory) {
    return candidates;
}

}
```



Step4: Write Migration Scripts.

Deploy Token (`1_deploy_token.js`)

Create a new file `1_deploy_token.js` in the `migrations` directory:

```
const MyToken = artifacts.require("MyToken");
```

```
module.exports = function (deployer) {
```

```
    deployer.deploy(MyToken, web3.utils.toWei('1000000', 'ether')) // 1,000,000 tokens  
};
```

```

const MyToken = artifacts.require("MyToken");
module.exports = function (deployer) {
  deployer.deploy(MyToken, web3.utils.toWei('1000000', 'ether')); // 1,000,000 tokens
};

```

Deploy Voting Contract (`2_deploy_voting.js`)

Create a new file `2_deploy_voting.js` in the `migrations` directory:

```
const MyToken = artifacts.require("MyToken");
```

```
const Voting = artifacts.require("Voting");
```

```
module.exports = async function (deployer, network, accounts) {
  const token = await MyToken.deployed();
  const candidates = [accounts[1], accounts[2], accounts[3]];
  await deployer.deploy(Voting, token.address, candidates);
};
```

```

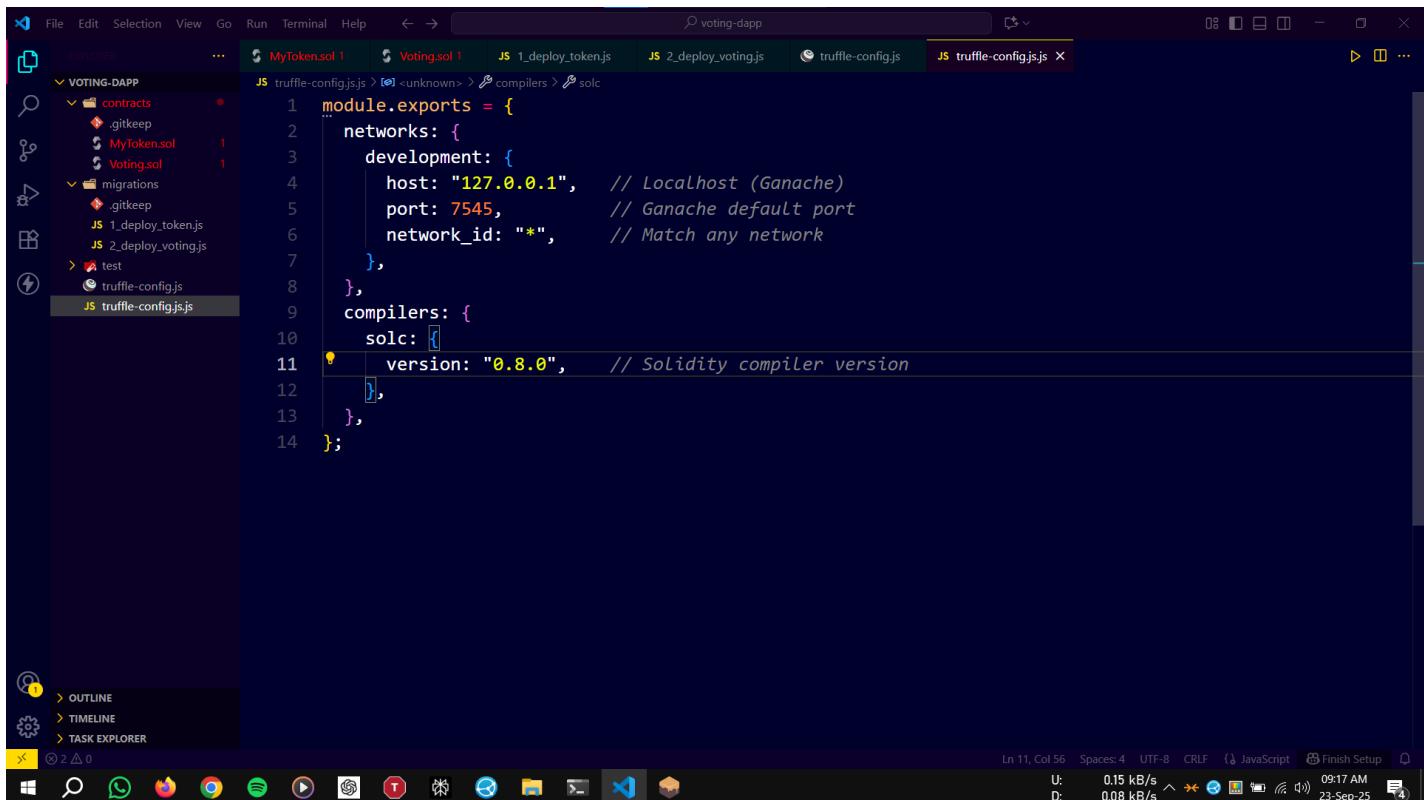
const MyToken = artifacts.require("MyToken");
const Voting = artifacts.require("Voting");
module.exports = async function (deployer, network, accounts) {
  const token = await MyToken.deployed();
  const candidates = [accounts[1], accounts[2], accounts[3]];
  await deployer.deploy(Voting, token.address, candidates);
};

```

Step5: Configure Truffle.

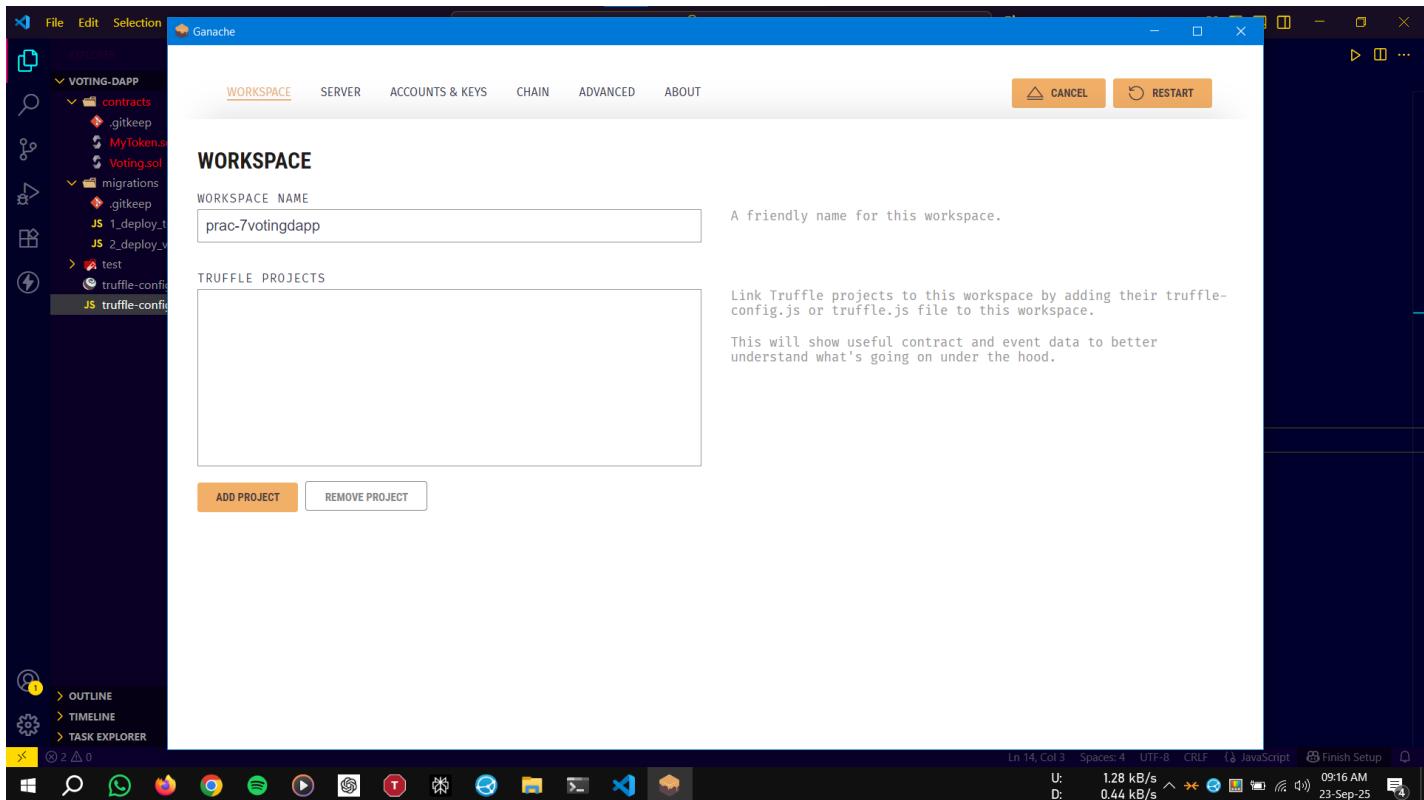
Ensure `truffle-config.js` is set up to use the local Ganache network:

```
module.exports = {  
  networks: {  
    development: {  
      host: "127.0.0.1", // Localhost (Ganache)  
      port: 7545, // Ganache default port  
      network_id: "*", // Match any network  
    },  
    },  
  compilers: {  
    solc: {  
      version: "0.8.0", // Solidity compiler version  
    },  
    },  
};
```



Step6: Start Ganache

Open Ganache and start a new workspace. Ensure it's running on port.



Step7: Deploy the Contracts

Run the following command to deploy your contracts

truffle migrate --network development

```
voting-dapp
JS oldtruffle-config.js > 0x <unknown> > compilers > solc > version
49 module.exports = {
101 // Set default mocha options here, use special reporters, etc.
102 mocha: {
103 // timeout: 100000
104 },
105
106 // Configure your compilers
107 compilers: {
108 solc: {
109 version: "0.8.0", // Fetch exact version from solc-bin (default: truffle's version)
110 // docker: true // Use "0.5.1" you've installed locally with docker (default: false)
111 }
112 }

> Compiling @openzeppelin\contracts\token\ERC20\ERC20.sol
> Compiling @openzeppelin\contracts\token\ERC20\IERC20.sol
> Compiling .\contracts\MyToken.sol
> Compiling .\contracts\Voting.sol

added 1 package in 2s
PS Z:\Tools for class\bcws-all-files\voting-dapp> truffle compile --all

Compiling your contracts...
=====
> Compiling @openzeppelin\contracts\token\ERC20\ERC20.sol
> Compiling @openzeppelin\contracts\token\ERC20\IERC20.sol
> Compiling .\contracts\MyToken.sol
> Compiling .\contracts\Voting.sol
> Compiling .\contracts\MyToken.sol
> Compiling .\contracts\Voting.sol
```

The screenshot shows the VS Code interface with the 'TERMINAL' tab active. The terminal window displays the output of the 'truffle migrate --network development' command. It shows the deployment of the 'MyToken.sol' and 'Voting.sol' contracts to the 'development' network. The bottom status bar shows network activity: U: 0.00 kB/s, D: 0.08 kB/s, and a timestamp of 09:21 AM on 23-Sep-25.

Step8: Create Frontend

Create index.html file:

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Voting DApp</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <h1>Token-Based Voting System</h1>
    <div class="vote-form">
      <input type="text" id="candidateAddress" placeholder="Candidate Address">
      <input type="number" id="voteAmount" placeholder="Amount">
      <button id="voteButton">Vote</button>
    </div>
    <div class="results">
      <h2>Vote Counts</h2>
      <ul id="voteList"></ul>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
    <script src="app.js"></script>
  </body>
</html>
```

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Voting DApp</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>Token-Based Voting System</h1>
<div class="vote-form">
<input type="text" id="candidateAddress" placeholder="Candidate Address">
<input type="number" id="voteAmount" placeholder="Amount">
<button id="voteButton">Vote</button>
</div>
<div class="results">
<h2>Vote Counts</h2>
<ul id="voteList"></ul>
</div>
<script src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
<script src="app.js"></script>
</body>
</html>

```

Create app.js file:

```

let web3, votingContract, tokenContract;

const votingContractAddress = "YOUR_VOTING_CONTRACT_ADDRESS";
const tokenContractAddress = "YOUR_TOKEN_CONTRACT_ADDRESS";
const votingContractABI = [ /* ABI from Truffle */ ];
const tokenContractABI = [ /* ABI from Truffle */ ];

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

    tokenContract = new web3.eth.Contract(tokenContractABI, tokenContractAddress);
    votingContract = new web3.eth.Contract(votingContractABI, votingContractAddress);

    document.getElementById("voteButton").addEventListener("click", async () => {
      const candidate = document.getElementById("candidateAddress").value;
      const amount = document.getElementById("voteAmount").value;
      const accounts = await web3.eth.getAccounts();
    });
  }
});

```

```
await tokenContract.methods.approve(votingContractAddress, amount).send({ from: accounts[0] });

await votingContract.methods.vote(candidate, amount).send({ from: accounts[0] });

updateVoteCounts();

});

updateVoteCounts();
} else {
  alert("MetaMask not detected. Please install MetaMask.");
}
};

async function updateVoteCounts() {
  const candidates = await votingContract.methods.get Candidates().call();
  const voteList = document.getElementById("voteList");
  voteList.innerHTML = "";
  for (let i = 0; i < candidates.length; i++) {
    const votes = await votingContract.methods.getVotes(candidates[i]).call();
    const listItem = document.createElement("li");
    listItem.textContent = `${candidates[i]}: ${votes} tokens`;
    voteList.appendChild(listItem);
  }
}
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** On the left, it displays the project structure under "VOTING-DAPP". The "contracts" folder contains "MyToken.sol" and "Voting.sol". The "migrations" folder contains "1_deploy_token.js" and "2_deploy_voting.js". Other files include "app.js", "index.html", "oldtruffle-config.js", "package-lock.json", "package.json", and "truffle-config.js".
- Code Editor:** The main area shows the content of "app.js". The code uses Web3.js to interact with Ethereum contracts. It includes logic for approving tokens and voting.
- Status Bar:** At the bottom, it shows "Ln 41, Col 2" and "Spaces: 4" with "CRLF" encoding. It also indicates the file is a "JavaScript" file and has been "Finish Setup".
- Bottom Icons:** Standard VS Code icons for file operations like Open, Save, Find, and Run.

```
let web3, votingContract, tokenContract;
const votingContractAddress = "YOUR_VOTING_CONTRACT_ADDRESS";
const tokenContractAddress = "YOUR_TOKEN_CONTRACT_ADDRESS";
const votingContractABI = [ /* ABI from Truffle */ ];
const tokenContractABI = [ /* ABI from Truffle */ ];

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

    tokenContract = new web3.eth.Contract(tokenContractABI, tokenContractAddress);
    votingContract = new web3.eth.Contract(votingContractABI, votingContractAddress);

    document.getElementById("voteButton").addEventListener("click", async () => {
      const candidate = document.getElementById("candidateAddress").value;
      const amount = document.getElementById("voteAmount").value;
      const accounts = await web3.eth.getAccounts();

      await tokenContract.methods.approve(votingContractAddress, amount).send({ from: accounts[0] });
      await votingContract.methods.vote(candidate, amount).send({ from: accounts[0] });
      updateVoteCounts();
    });
  }
});

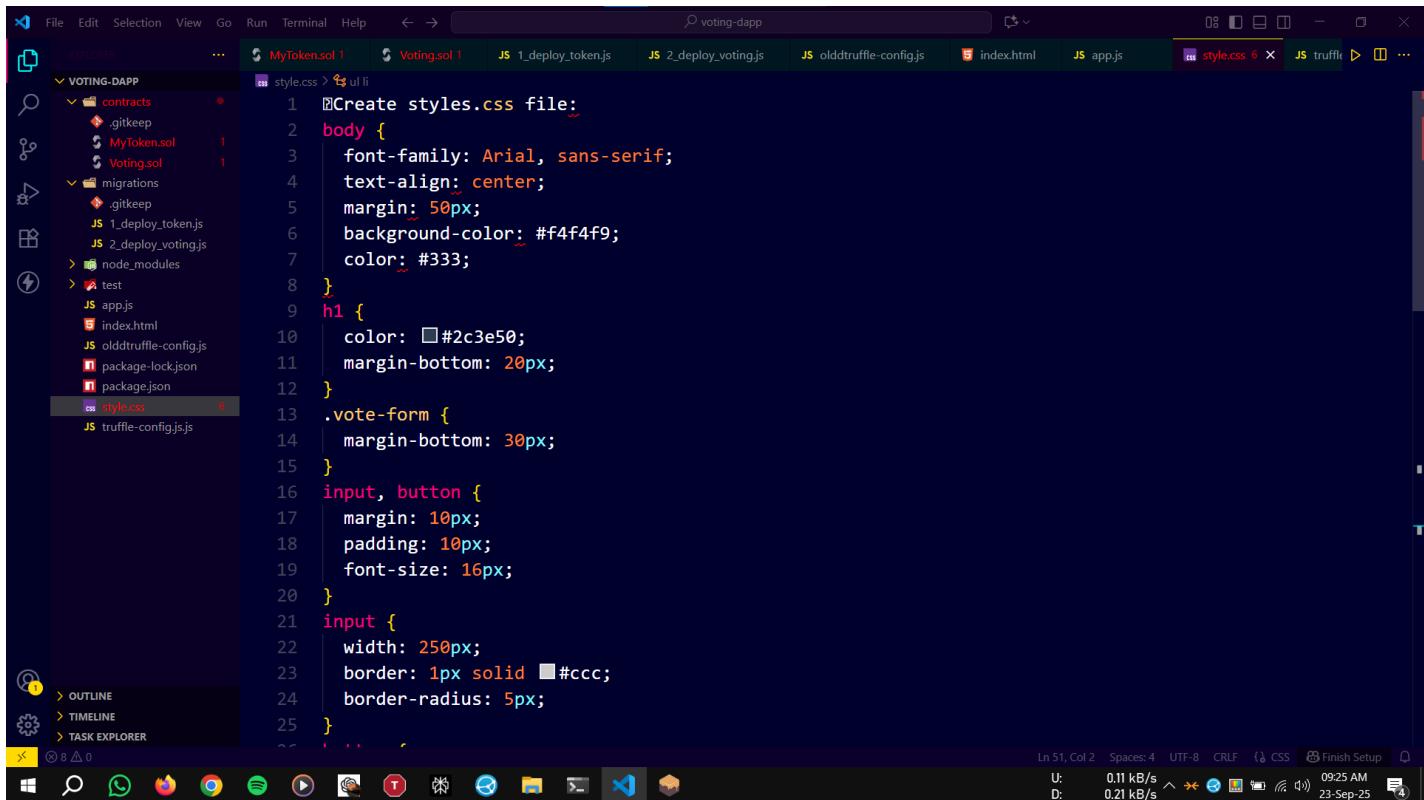
updateVoteCounts();
```

Create styles.css file:

```
body {  
    font-family: Arial, sans-serif;  
    text-align: center;  
    margin: 50px;  
    background-color: #f4f4f9;  
    color: #333;  
}  
  
h1 {  
    color: #2c3e50;  
    margin-bottom: 20px;  
}  
  
.vote-form {  
    margin-bottom: 30px;  
}  
  
input, button {  
    margin: 10px;  
    padding: 10px;  
    font-size: 16px;  
}
```

2203031260184

```
input {  
    width: 250px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
}  
  
button {  
    background-color: #3498db;  
    color: white;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
    transition: background 0.3s ease;  
}  
  
button:hover {  
    background-color: #2980b9;  
}  
  
.results {  
    margin-top: 30px;  
}  
  
ul {  
    list-style-type: none;  
    padding: 0;  
}  
  
ul li {  
    background: #ecf0f1;  
    margin: 8px auto;  
    padding: 12px;  
    border-radius: 5px;  
    width: 300px;  
    text-align: center;  
}
```



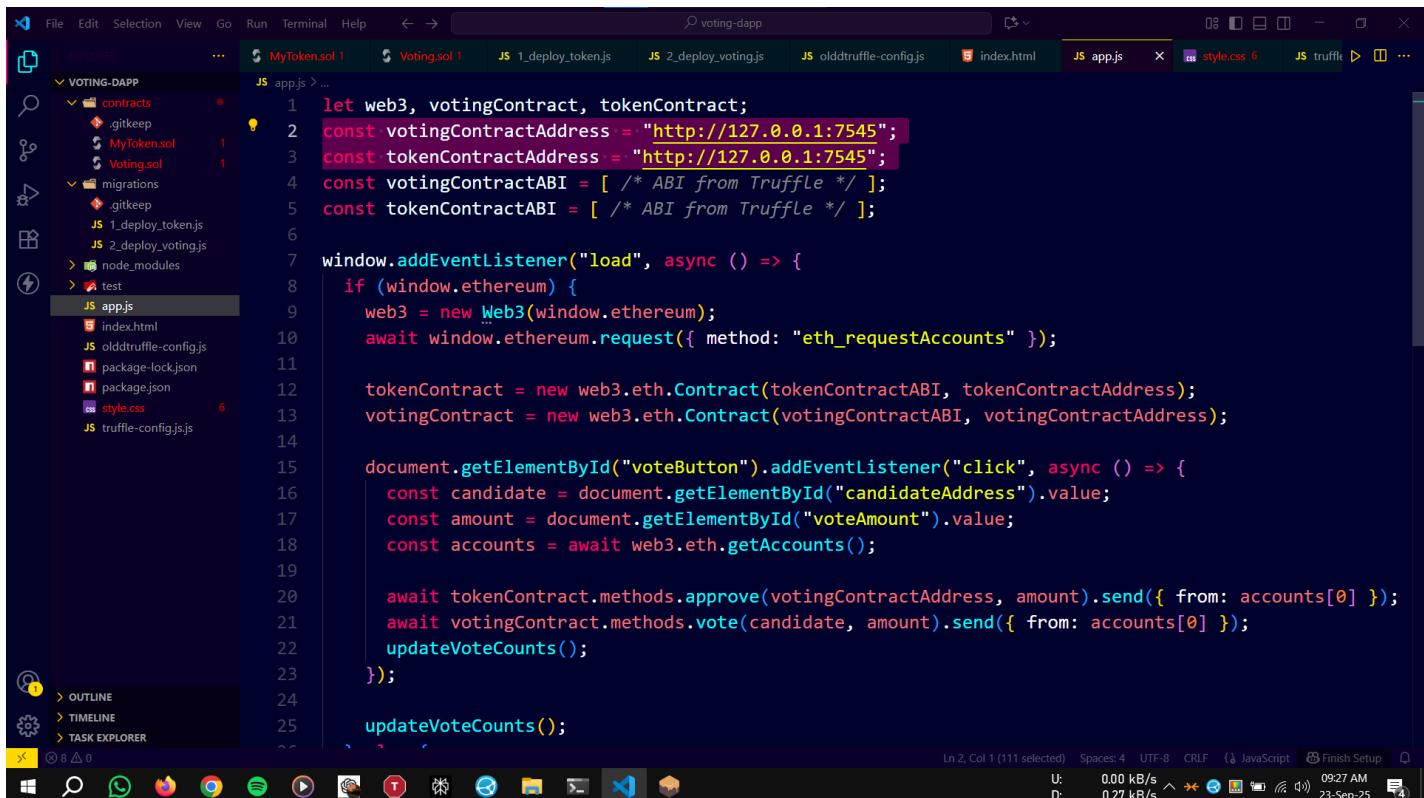
```

Create styles.css file:
body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 50px;
    background-color: #f4f4f9;
    color: #333;
}
h1 {
    color: #2c3e50;
    margin-bottom: 20px;
}
.vote-form {
    margin-bottom: 30px;
}
input, button {
    margin: 10px;
    padding: 10px;
    font-size: 16px;
}
input {
    width: 250px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

```

Step9: Get Contract ABIs

After deploying the contracts, retrieve the ABI from `build/contracts` for both `voting` and `mytoken` contracts. Replace the placeholders in `app.js` with the actual ABIs and contract addresses.



```

let web3, votingContract, tokenContract;
const votingContractAddress = "http://127.0.0.1:7545";
const tokenContractAddress = "http://127.0.0.1:7545";
const votingContractABI = [ /* ABI from Truffle */ ];
const tokenContractABI = [ /* ABI from Truffle */ ];

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

        tokenContract = new web3.eth.Contract(tokenContractABI, tokenContractAddress);
        votingContract = new web3.eth.Contract(votingContractABI, votingContractAddress);

        document.getElementById("voteButton").addEventListener("click", async () => {
            const candidate = document.getElementById("candidateAddress").value;
            const amount = document.getElementById("voteAmount").value;
            const accounts = await web3.eth.getAccounts();

            await tokenContract.methods.approve(votingContractAddress, amount).send({ from: accounts[0] });
            await votingContract.methods.vote(candidate, amount).send({ from: accounts[0] });
            updateVoteCounts();
        });
        updateVoteCounts();
    }
});

```

Step10: Run the Frontend

Use `http-server` to serve your frontend

http-server

The screenshot shows the VS Code interface with the 'voting-dapp' project open. The Explorer sidebar shows files like MyToken.sol, Voting.sol, migrations, node_modules, test, app.js, index.html, oldtruffle-config.js, package-lock.json, package.json, style.css, and truffle-config.js.js. The terminal tab displays the command 'PS Z:\Tools\for class\bcws-all-files\voting-dapp> http-server' followed by the server's configuration and the URL 'http://192.168.56.2:8080'.

```

let web3, votingContract, tokenContract;
const votingContractAddress = "http://127.0.0.1:7545";
const tokenContractAddress = "http://127.0.0.1:7545";
const votingContractABI = [ /* ABI from Truffle */ ];
const tokenContractABI = [ /* ABI from Truffle */ ];

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

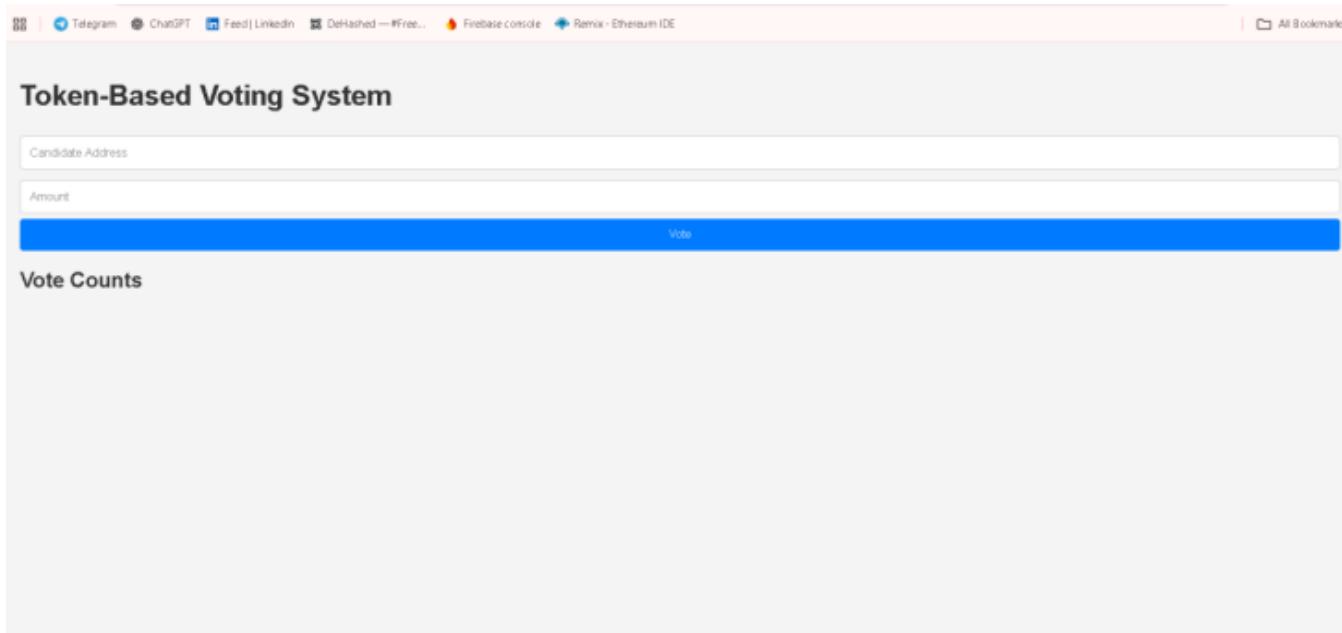
spelling of the name, or if a path was included, verify that the path is correct and try again.
PS Z:\Tools\for class\bcws-all-files\voting-dapp> http-server
○ Starting up http-server, serving .
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://192.168.56.2:8080

```

Step11: Open the browser and navigate to `http://127.0.0.1:8081`. Connect MetaMask and interact with the voting dApp.



Conclusion:

We have successfully deployed Voting DApp.

Practical - 8

Aim: Securing Decentralized Identity

1. Exploring decentralized identity solutions
2. Implementing secure identity management on the blockchain

Step1: Setup

Install Node.js, npm, Truffle, Ganache, MetaMask, and http-server.

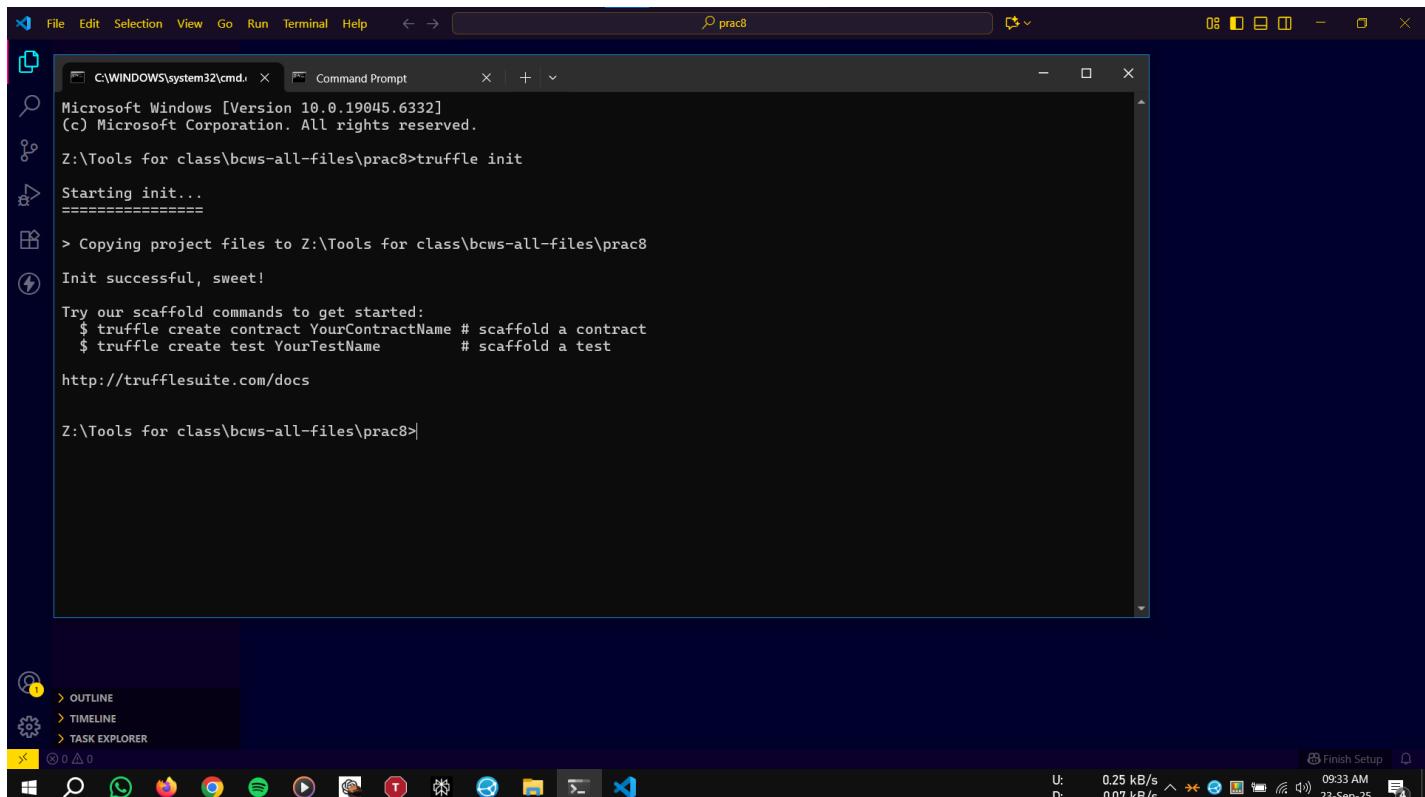
Create project:

```
mkdir prac8
```

```
cd prac8
```

Initialize truffle :

```
truffle init
```



```
C:\WINDOWS\system32\cmd. x Command Prompt x + v
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

Z:\Tools for class\bcws-all-files\prac8>truffle init

Starting init...
=====
> Copying project files to Z:\Tools for class\bcws-all-files\prac8

Init successful, sweet!

Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName           # scaffold a test

http://trufflesuite.com/docs

Z:\Tools for class\bcws-all-files\prac8>
```

Step2: Smart Contract

Create DIDRegistry.sol in contracts directory.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract DIDRegistry {
    struct DIDDocument {
        address controller;      // DID controller
```

```
string metadata;          // metadata pointer (IPFS CID, JSON URL, etc.)
uint256 updatedAt;       // last update timestamp
}

mapping(bytes32 => DIDDocument) private docs;
event DIDRegistered(
    bytes32 indexed didHash,
    string did,
    address indexed controller,
    string metadata
);
event DIDUpdated(
    bytes32 indexed didHash,
    string did,
    address indexed controller,
    string metadata
);
event ControllerTransferred(
    bytes32 indexed didHash,
    string did,
    address indexed oldController,
    address indexed newController
);
modifier onlyController(bytes32 h) {
    require(docs[h].controller == msg.sender, "Caller is not controller");
    -
}
function registerDID(string calldata did, string calldata metadata) external {
    bytes32 h = keccak256(abi.encodePacked(did));
    require(docs[h].controller == address(0), "DID already registered");
    docs[h] = DIDDocument({
        controller: msg.sender,
        metadata: metadata,
        updatedAt: block.timestamp
    });
    emit DIDRegistered(h, did, msg.sender, metadata);
}
```

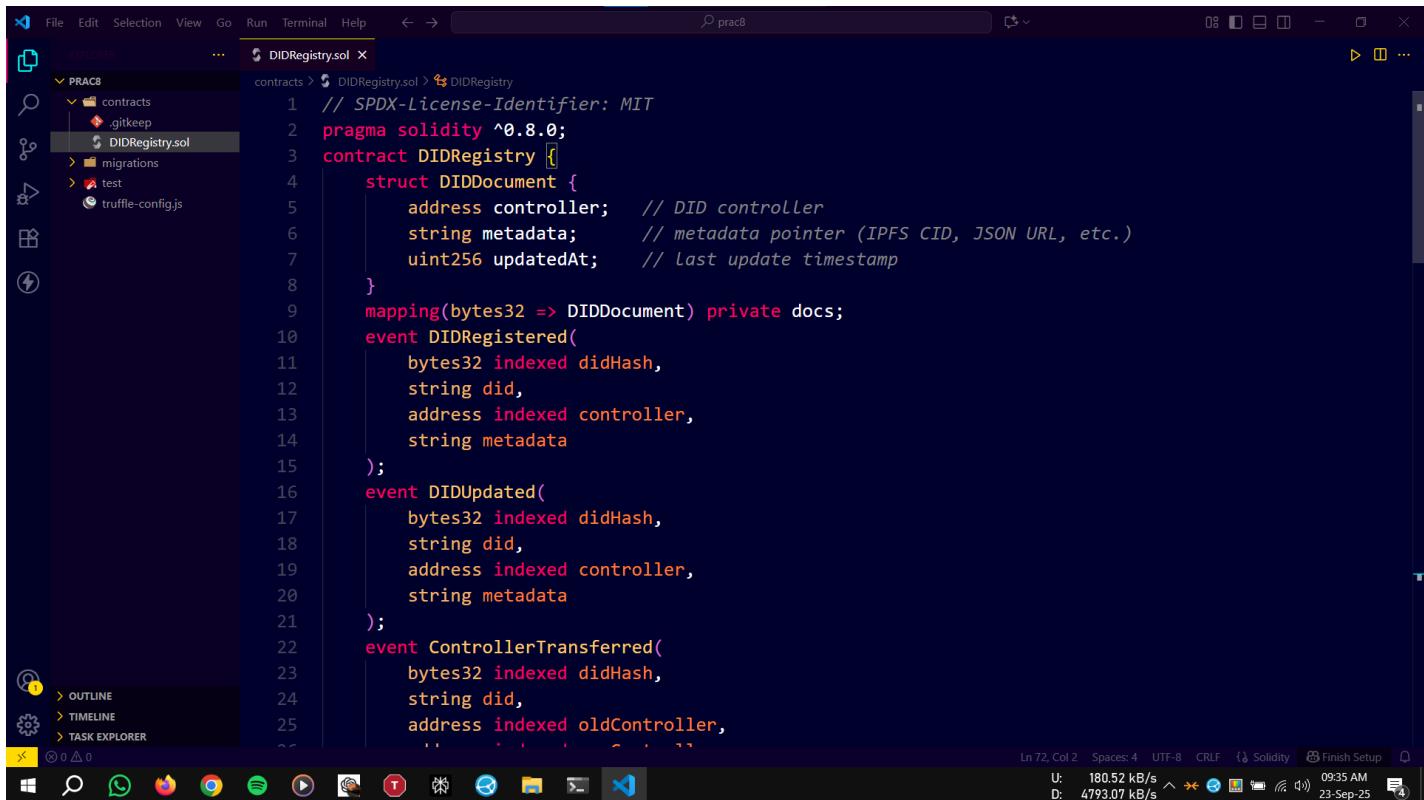
```
}

function updateMetadata(string calldata did, string calldata metadata) external {
    bytes32 h = keccak256(abi.encodePacked(did));
    require(docs[h].controller != address(0), "DID not registered");
    require(docs[h].controller == msg.sender, "Caller is not controller");
    docs[h].metadata = metadata;
    docs[h].updatedAt = block.timestamp;
    emit DIDUpdated(h, did, msg.sender, metadata);
}

function transferController(string calldata did, address newController) external {
    bytes32 h = keccak256(abi.encodePacked(did));
    require(docs[h].controller != address(0), "DID not registered");
    require(docs[h].controller == msg.sender, "Caller is not controller");
    address oldController = docs[h].controller;
    docs[h].controller = newController;
    docs[h].updatedAt = block.timestamp;
    emit ControllerTransferred(h, did, oldController, newController);
}

function getDID(string calldata did) external view returns (
    address controller,
    string memory metadata,
    uint256 updatedAt
) {
    bytes32 h = keccak256(abi.encodePacked(did));
    DIDDocument memory d = docs[h];
    return (d.controller, d.metadata, d.updatedAt);
}

function controllerOf(string calldata did) external view returns (address) {
    bytes32 h = keccak256(abi.encodePacked(did));
    return docs[h].controller;
}
```



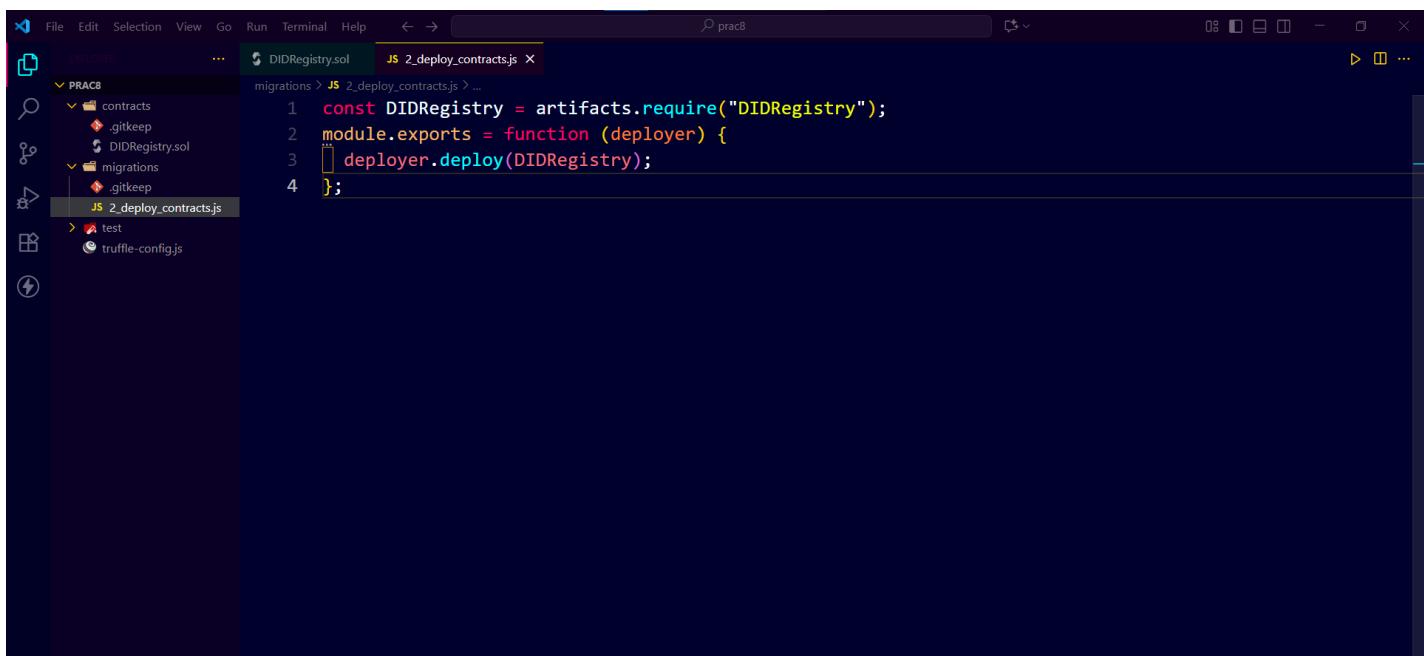
The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under 'PRAC8' with files like 'contracts/DIDRegistry.sol', '.gitkeep', 'migrations/JS_2_deploy_contracts.js', 'test', and 'truffle-config.js'.
- Code Editor:** Displays the Solidity code for 'DIDRegistry.sol'. The code defines a struct 'DIDDocument' with fields: address controller, string metadata, and uint256 updatedAt. It includes three events: DIDRegistered, DIDUpdated, and ControllerTransferred, each with parameters related to the DIDDocument.
- Bottom Status Bar:** Shows file statistics: Line 72, Column 2; Spaces: 4; UTF-8; CRLF; and network speeds: U: 180.52 kB/s, D: 4793.07 kB/s. It also displays the date and time: 09:35 AM, 23-Sep-25.

Create 2_deploy_contracts.js in migrations folder:

```
const DIDRegistry = artifacts.require("DIDRegistry");

module.exports = function (deployer) {
  deployer.deploy(DIDRegistry);
};
```

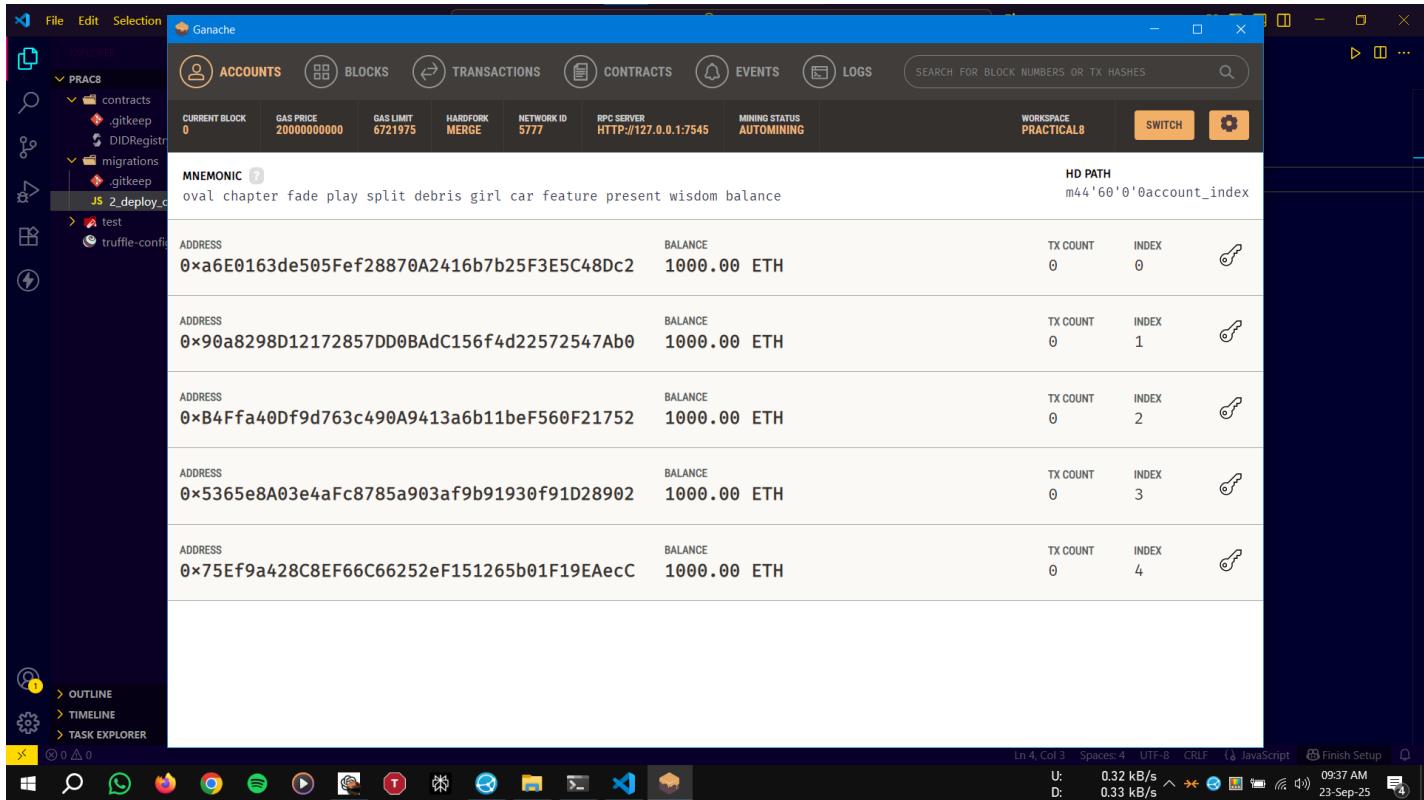


The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under 'PRAC8' with files like 'contracts/DIDRegistry.sol', '.gitkeep', 'migrations/JS_2_deploy_contracts.js', 'test', and 'truffle-config.js'.
- Code Editor:** Displays the JavaScript code for 'JS_2_deploy_contracts.js'. It uses the 'artifacts.require' method to import 'DIDRegistry' and then defines a module export function that calls 'deployer.deploy(DIDRegistry)'.

Step3: Deploy

Start Ganache.



Edit truffle-config.js according to Ganache host/port.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*",
    },
    compilers: {
      solc: {
        version: "0.8.0",
      },
    },
  };
};
```

```

1 module.exports = {
2   networks: {
3     development: {
4       host: "127.0.0.1",
5       port: 7545,
6       network_id: "*",
7     },
8   },
9   compilers: {
10     solc: {
11       version: "0.8.0",
12     },
13   },
14 };

```

Compile & deploy:

truffle compile

truffle migrate --reset

```

PS Z:\Tools for class\bcws-all-files\prac8> truffle compile

Compiling your contracts...
=====
> Compiling ./contracts\DidRegistry.sol
> Artifacts written to Z:\Tools for class\bcws-all-files\prac8\build\contracts
> Compiled successfully using:
- solc: 0.8.21+commit.d9974bed.Emscripten clang

PS Z:\Tools for class\bcws-all-files\prac8> truffle migrate --reset

Compiling your contracts...
=====
> Compiling ./contracts\DidRegistry.sol
> Artifacts written to Z:\Tools for class\bcws-all-files\prac8\build\contracts
> Compiled successfully using:
- solc: 0.8.21+commit.d9974bed.Emscripten clang

Starting migrations...
=====
> Network name: ganache
> Network id: 5777
> Block gas limit: 6721975 (0x6691b7)

```

Step4: Frontend

Create 3 files in project root:

index.html (form to register/retrieve DID)

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta charset="utf-8" />
```

```
  <title>DID Management</title>
```

```
  <link rel="stylesheet" href="styles.css" />
```

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

```
</head>
```

```
<body>
```

```
  <div class="container">
```

```
    <h1>Decentralized Identity Management</h1>
```

```
    <section>
```

```
      <h2>Register DID</h2>
```

```
      <form id="register-form">
```

```
        <input id="did" placeholder="Enter DID (e.g., did:example:123)" required />
```

```
        <input id="metadata" placeholder="Enter metadata (e.g., ipfs://...)" required />
```

```
        <button type="submit">Register</button>
```

```
      </form>
```

```
    </section>
```

```
    <h2>Retrieve DID</h2>
```

```
    <form id="retrieve-form">
```

```
      <input id="retrieveDid" placeholder="Enter DID" required />
```

```
      <button type="submit">Retrieve</button>
```

```
    </form>
```

```
    <div id="did-info"></div>
```

```
  </section>
```

```
<section>
    <h2>Contract Info</h2>
    <p>Contract Address: <span id="http://127.0.0.1:7545"></span></p>
</section>
</div>

<script src="https://cdn.jsdelivr.net/npm/web3/dist/web3.min.js"></script>
<script src="app.js"></script>
</body>
</html>
```

The screenshot shows the Visual Studio Code interface with the file 'index.html' open in the editor. The code is written in HTML and includes sections for contract info and registration forms.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>DID Management</title>
    <link rel="stylesheet" href="styles.css" />
    <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
    <div class="container">
        <h1>Decentralized Identity Management</h1>

        <section>
            <h2>Register DID</h2>
            <form id="register-form">
                <input id="did" placeholder="Enter DID (e.g., did:example:123)" required />
                <input id="metadata" placeholder="Enter metadata (e.g., ipfs://...)" required />
                <button type="submit">Register</button>
            </form>
        </section>
    </div>
</body>

```

styles.css (basic styling)

```
body {
    font-family: Arial, sans-serif;
    background: #f4f4f4;
    margin: 0;
    padding: 20px;
}
```

```
.container {  
    max-width: 700px;  
    margin: auto;  
    background: #fff;  
    padding: 20px;  
    border-radius: 8px;  
    box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.1);  
}
```

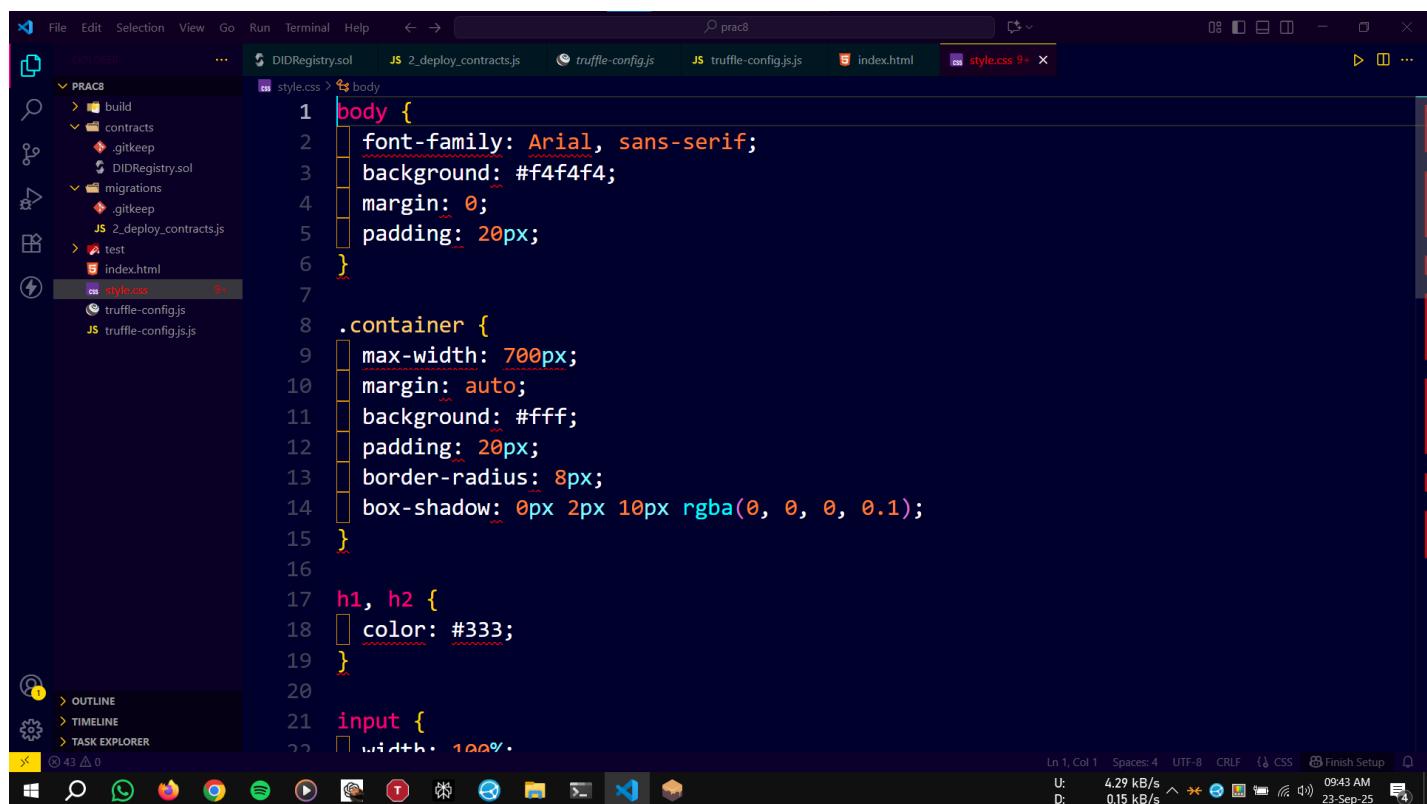
```
h1, h2 {  
    color: #333;  
}
```

```
input {  
    width: 100%;  
    padding: 10px;  
    margin: 6px 0;  
    box-sizing: border-box;  
    border: 1px solid #ccc;  
    border-radius: 4px;  
}
```

```
button {  
    width: 100%;  
    padding: 10px;  
    background: #007bff;  
    color: #fff;  
    border: none;  
    border-radius: 4px;  
    cursor: pointer;  
}
```

```
button:hover {  
    background: #0056b3;  
}
```

```
#did-info {  
    margin-top: 12px;  
    padding: 10px;  
    background: #f1f1f1;  
    border-radius: 4px;  
}
```



```
body {  
    font-family: Arial, sans-serif;  
    background: #f4f4f4;  
    margin: 0;  
    padding: 20px;  
}  
  
.container {  
    max-width: 700px;  
    margin: auto;  
    background: #fff;  
    padding: 20px;  
    border-radius: 8px;  
    box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.1);  
}  
  
h1, h2 {  
    color: #333;  
}  
  
input {  
    width: 100%;  
}
```

app.js (connect to contract with Web3.js)

```
// Replace with your deployed contract address after migration
```

```
const CONTRACT_ADDRESS = "http://127.0.0.1:8545";
```

```
// Replace with the ABI from build/contracts/DIDRegistry.json
```

```
const DIDRegistryABI = [
```

```
// PASTE FULL ABI ARRAY HERE
```

```
};

let web3;
let accounts;
let contract;

async function init() {
  if (window.ethereum) {
    web3 = new Web3(window.ethereum);
    await window.ethereum.request({ method: "eth_requestAccounts" });
  } else {
    web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));
  }

  accounts = await web3.eth.getAccounts();
  contract = new web3.eth.Contract(DIDRegistryABI, CONTRACT_ADDRESS);

  document.getElementById("contract-address").innerText = CONTRACT_ADDRESS;
}

document.getElementById("register-form").onsubmit = async (e) => {
  e.preventDefault();
  const did = document.getElementById("did").value.trim();
  const metadata = document.getElementById("metadata").value.trim();

  try {
    await contract.methods.registerDID(did, metadata).send({ from: accounts[0] });
    alert("DID registered successfully!");
  } catch (err) {
    console.error(err);
    alert(`Registration failed: ${err.message}`);
  }
}
```

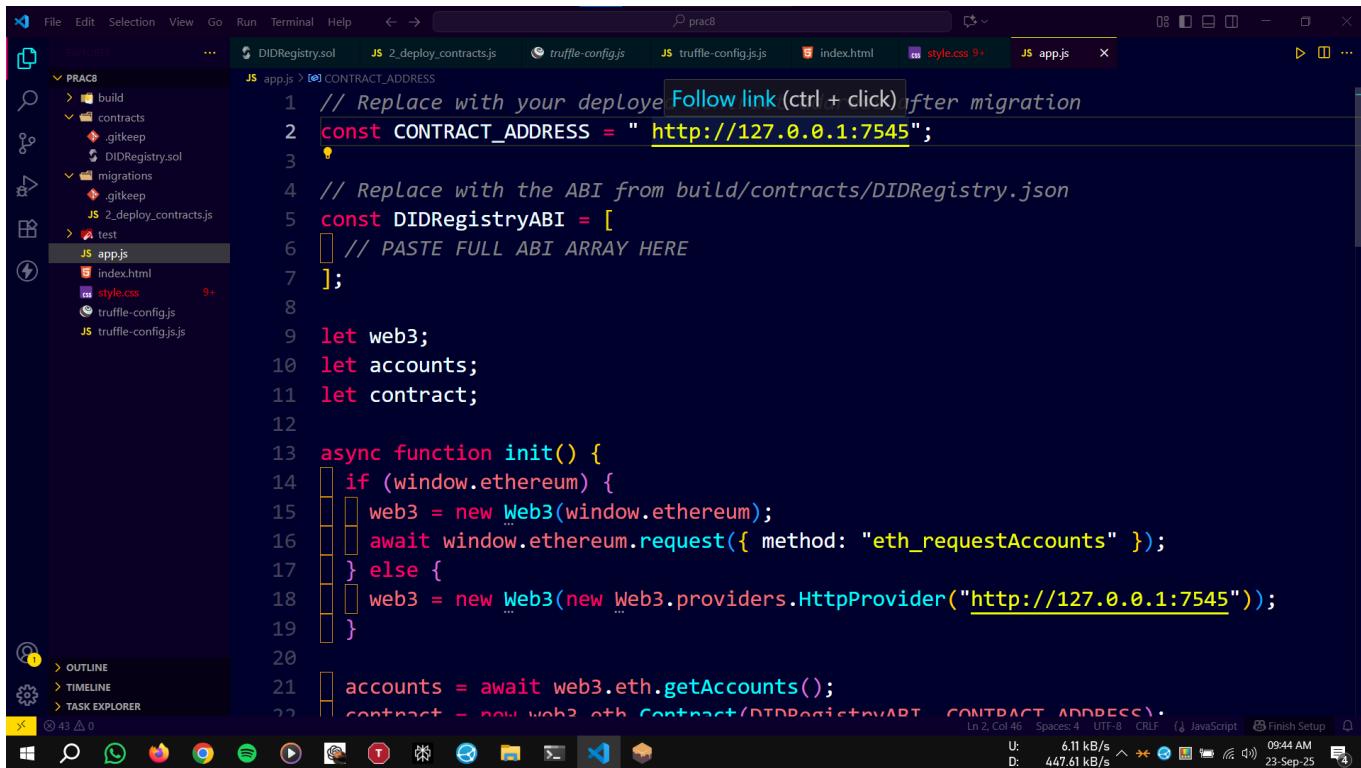
```
};

document.getElementById("retrieve-form").onsubmit = async (e) => {
  e.preventDefault();
  const did = document.getElementById("retrieveDid").value.trim();

  try {
    const result = await contract.methods.getDID(did).call();
    const controller = result[0];
    const metadata = result[1];
    const updatedAt = result[2];
    const time = updatedAt > 0 ? new Date(updatedAt * 1000).toLocaleString() : "Never";

    document.getElementById("did-info").innerHTML = `
      <p><strong>DID:</strong> ${did}</p>
      <p><strong>Controller:</strong> ${controller}</p>
      <p><strong>Metadata:</strong> ${metadata}</p>
      <p><strong>Updated At:</strong> ${time}</p>
    `;
  } catch (err) {
    console.error(err);
    alert("Retrieval failed: " + err.message);
  }
};

window.addEventListener("load", init);
```



```

1 // Replace with your deployed contract address after migration
2 const CONTRACT_ADDRESS = "http://127.0.0.1:7545";
3
4 // Replace with the ABI from build/contracts/DIDRegistry.json
5 const DIDRegistryABI = [
6   // PASTE FULL ABI ARRAY HERE
7 ];
8
9 let web3;
10 let accounts;
11 let contract;
12
13 async function init() {
14   if (window.ethereum) {
15     web3 = new Web3(window.ethereum);
16     await window.ethereum.request({ method: "eth_requestAccounts" });
17   } else {
18     web3 = new Web3(new Web3.providers.HttpProvider("http://127.0.0.1:7545"));
19   }
20
21   accounts = await web3.eth.getAccounts();
22   contract = new web3.eth.Contract(DIDRegistryABI, CONTRACT_ADDRESS);

```

Add the address of smart contract into app.js obtained from truffle migrate command from the terminal

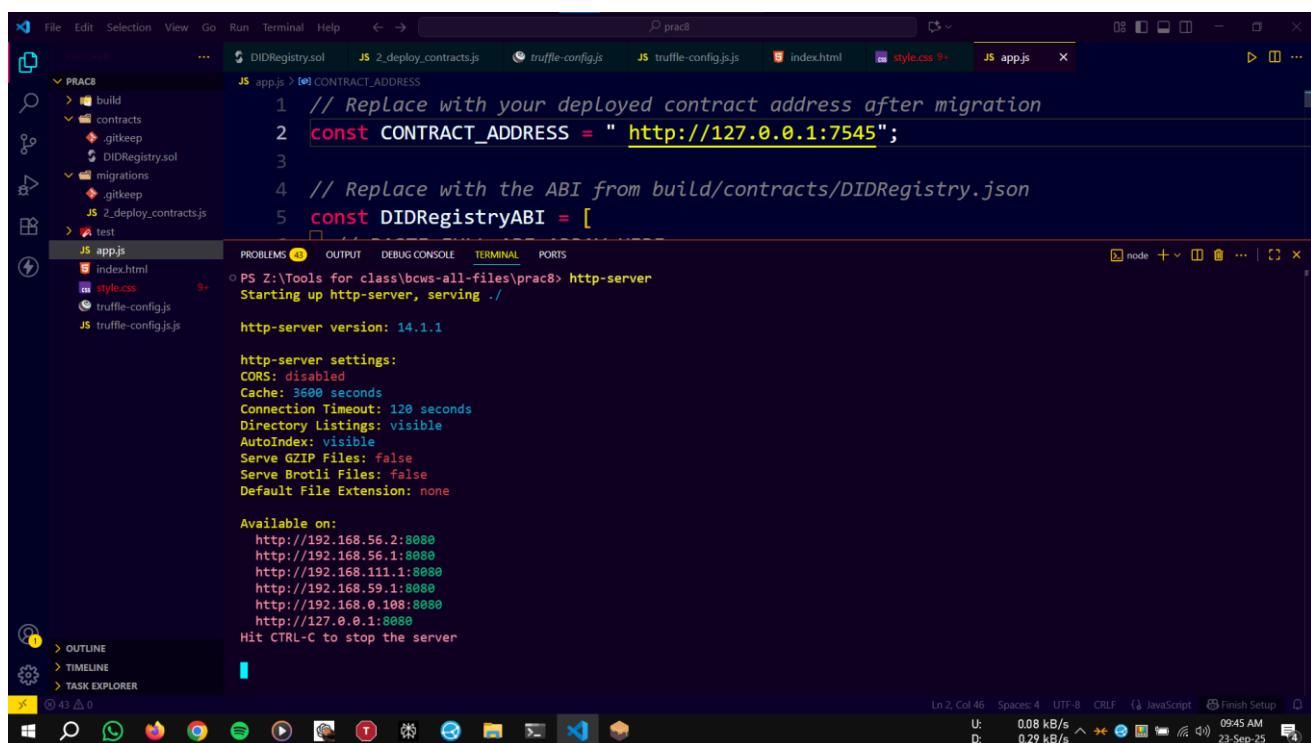
Step5: Run App

Start local server:

http-server

Open <http://127.0.0.1:8080> in browser.

Connect MetaMask → point it to Ganache network.



```

1 // Replace with your deployed contract address after migration
2 const CONTRACT_ADDRESS = "http://127.0.0.1:7545";
3
4 // Replace with the ABI from build/contracts/DIDRegistry.json
5 const DIDRegistryABI = [

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS Z:\Tools for class\bcws-all-files\prac8> http-server
Starting up http-server, serving .
http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
http://192.168.56.2:8080
http://192.168.56.1:8080
http://192.168.111.1:8080
http://192.168.59.1:8080
http://192.168.0.108:8080
http://127.0.0.1:8080
Hit CTRL-C to stop the server

```

Step6: Test

Use form to register a DID.

Use form to retrieve DID info.

Confirm success messages and on-chain data.

The screenshot shows a web application titled "Decentralized Identity Management". The interface is divided into three main sections:

- Register DID:** Contains two input fields: "Enter DID (e.g., did:example:123)" and "Enter metadata (e.g., ipfs://...)", followed by a blue "Register" button.
- Retrieve DID:** Contains one input field: "Enter DID", followed by a blue "Retrieve" button.
- Contract Info:** Displays the placeholder text "Contract Address:".

The browser's address bar shows the URL "127.0.0.1:8080". The top navigation bar includes links to Telegram, ChatGPT, LinkedIn, DeHashed, Firebase console, and Remix - Ethereum IDE, along with standard browser controls like back, forward, and search.

Practical - 9

Aim: Consensus Mechanisms and Security Implications

1. Understanding various consensus algorithms
2. Analyzing security implications of different consensus mechanisms
3. Hands-on exploration of consensus protocols

Theory -

In decentralized networks and blockchain systems, consensus mechanisms ensure that all participating nodes agree on a common state of the ledger without relying on a central authority. They are crucial for maintaining integrity, preventing fraud, and ensuring network security.

1. Consensus Algorithms

1.1 Proof of Work (PoW)

- **Mechanism:** Miners solve computational puzzles to validate transactions and produce new blocks.
- **Use Case:** Bitcoin and early cryptocurrencies.
- **Advantages:** Highly secure, decentralized, resistant to Sybil attacks.
- **Drawbacks:** Energy-intensive, low throughput (e.g., Bitcoin ~7 TPS).

1.2 Proof of Stake (PoS)

- **Mechanism:** Validators are chosen based on the tokens they lock as collateral.
- **Use Case:** Ethereum 2.0.
- **Advantages:** Energy-efficient, faster than PoW.
- **Drawbacks:** Risk of centralization; wealthy validators gain more influence.

1.3 Delegated Proof of Stake (DPoS)

- **Mechanism:** Token holders elect a small set of delegates to validate transactions.
- **Use Case:** EOS, Tron, Lisk.
- **Advantages:** High scalability and performance.
- **Drawbacks:** Risk of centralization and vulnerability to targeted DDoS attacks.

1.4 Practical Byzantine Fault Tolerance (PBFT)

- **Mechanism:** Nodes (replicas) achieve consensus by majority agreement, tolerating faulty or malicious actors.
- **Use Case:** Hyperledger Fabric.
- **Advantages:** Low transaction finality time, tolerant to node failures.
- **Drawbacks:** Communication overhead, scalability issues in large networks.

1.5 Proof of Authority (PoA)

- **Mechanism:** A limited number of trusted authorities validate blocks.
- **Use Case:** Private/consortium blockchains (e.g., VeChain).
- **Advantages:** Efficient, fast, suitable for permissioned systems.
- **Drawbacks:** Centralized by design, requires trust in validators.

2. Security Implications

2.1 Proof of Work (PoW)

- **51% Attack:** Possible if majority hash power is controlled by one entity.
- **Sybil Resistance:** High, due to cost of obtaining hash power.
- **Issue:** Enormous energy consumption; potential environmental and regulatory concerns.

2.2 Proof of Stake (PoS)

- **Staking Security:** Malicious validators risk losing funds (slashing).
- **Nothing-at-Stake Problem:** Validators may vote on multiple chains.
- **Risk:** Wealthy validators gain more power, reducing fairness.

2.3 Delegated Proof of Stake (DPoS)

- **Centralization:** Small set of delegates may monopolize control.
- **Vulnerability:** DDoS attacks against few validators can disrupt the network.

2.4 Byzantine Fault Tolerance (BFT) Protocols

- **Strength:** Tolerant as long as $<1/3$ of nodes are faulty.
- **Challenges:** High communication overhead and poor scalability in large public chains.

2.5 Proof of Authority (PoA)

- **Trust Model:** Relies on trusted authorities; compromised nodes can harm the system.
- **Centralization:** Security trade-off for efficiency.

3. Hands-on Exploration of Consensus Protocols

To better understand consensus mechanisms, we simulate two widely used protocols: Proof of Work (PoW) and Proof of Stake (PoS).

3.1 Proof of Work (PoW) Exercise

- **Tools:** Bitcoin Testnet, Mining Simulator (e.g., CGMiner, BFGMiner).
- **Steps:**
 1. Configure and connect a node on Bitcoin Testnet.
 2. Start mining blocks and track hash rate and energy usage.
 3. Attempt a double-spend attack in the test environment.

- **Observations:**

- Mining difficulty adjusts dynamically.
 - A 51% attack is highly impractical due to cost and computational requirements.

3.2 Proof of Stake (PoS) Exercise

- **Tools:** Ethereum 2.0 Testnet (Prysm, Lighthouse).

- **Steps:**

1. Set up a validator node on Ethereum 2.0 testnet.
2. Stake ETH (testnet tokens) to participate in validation.
3. Observe reward distribution and penalties for downtime.
4. Attempt malicious behavior and observe slashing.

- **Observations:**

- Energy requirements are minimal compared to PoW.
 - Validators must maintain consistent uptime to avoid penalties.

Practical - 10

Aim: Blockchain Security Audit and Best Practices

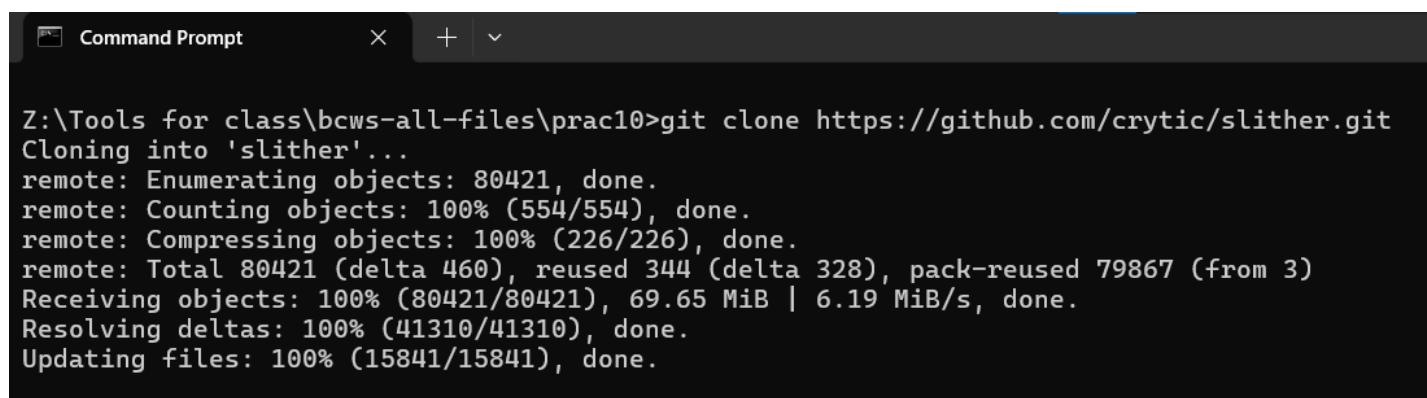
1. Conducting a security audit for a blockchain application
2. Implementing advanced security measures and best practices
3. Presentation and discussion of security findings

Tools Required -

1. Blockchain Development Framework: Truffle.
2. Local Blockchain Network: Ganache.
3. Smart Contract Development: Solidity.
4. Development Environment: VS Code.
5. Audit Tools: MythX, Slither.
6. Web Interface: HTML/CSS/JavaScript with MetaMask

Step1: Download slither from <https://github.com/crytic/slither?tab=readme-ov-file#how-to-install>

```
git clone https://github.com/crytic/slither.git
```



```
Z:\Tools for class\bcws-all-files\prac10>git clone https://github.com/crytic/slither.git
Cloning into 'slither'...
remote: Enumerating objects: 80421, done.
remote: Counting objects: 100% (554/554), done.
remote: Compressing objects: 100% (226/226), done.
remote: Total 80421 (delta 460), reused 344 (delta 328), pack-reused 79867 (from 3)
Receiving objects: 100% (80421/80421), 69.65 MiB | 6.19 MiB/s, done.
Resolving deltas: 100% (41310/41310), done.
Updating files: 100% (15841/15841), done.
```

Step2: In a slither folder create test.sol file with the vulnerable code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract MyContract {
    uint public balance;
    function deposit() public payable {
        balance += msg.value;
    }
}
```

```

function withdraw(uint amount) public {
    require(balance >= amount, "Not enough balance");
    balance -= amount;
    payable(msg.sender).transfer(amount);
}
}

```

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract MyContract {
    uint public balance;
    function deposit() public payable {
        balance += msg.value;
    }
    function withdraw(uint amount) public {
        require(balance >= amount, "Not enough balance");
        balance -= amount;
        payable(msg.sender).transfer(amount);
    }
}

```

Step3: Slither is a Python package, so you need Python + pip.

Make sure you have Python 3.8+ installed:

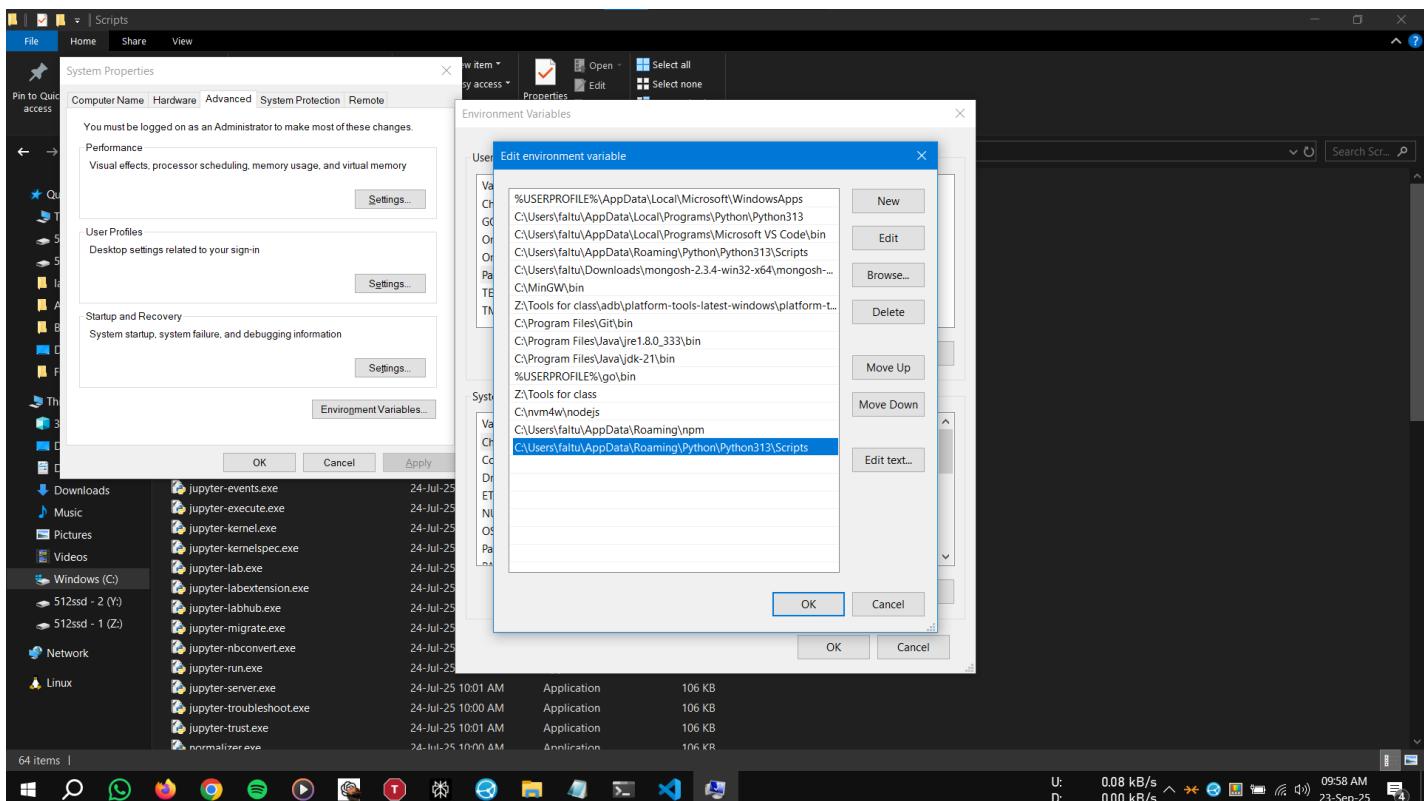
`python --version`

Install Slither globally:

`pip install slither-analyzer`

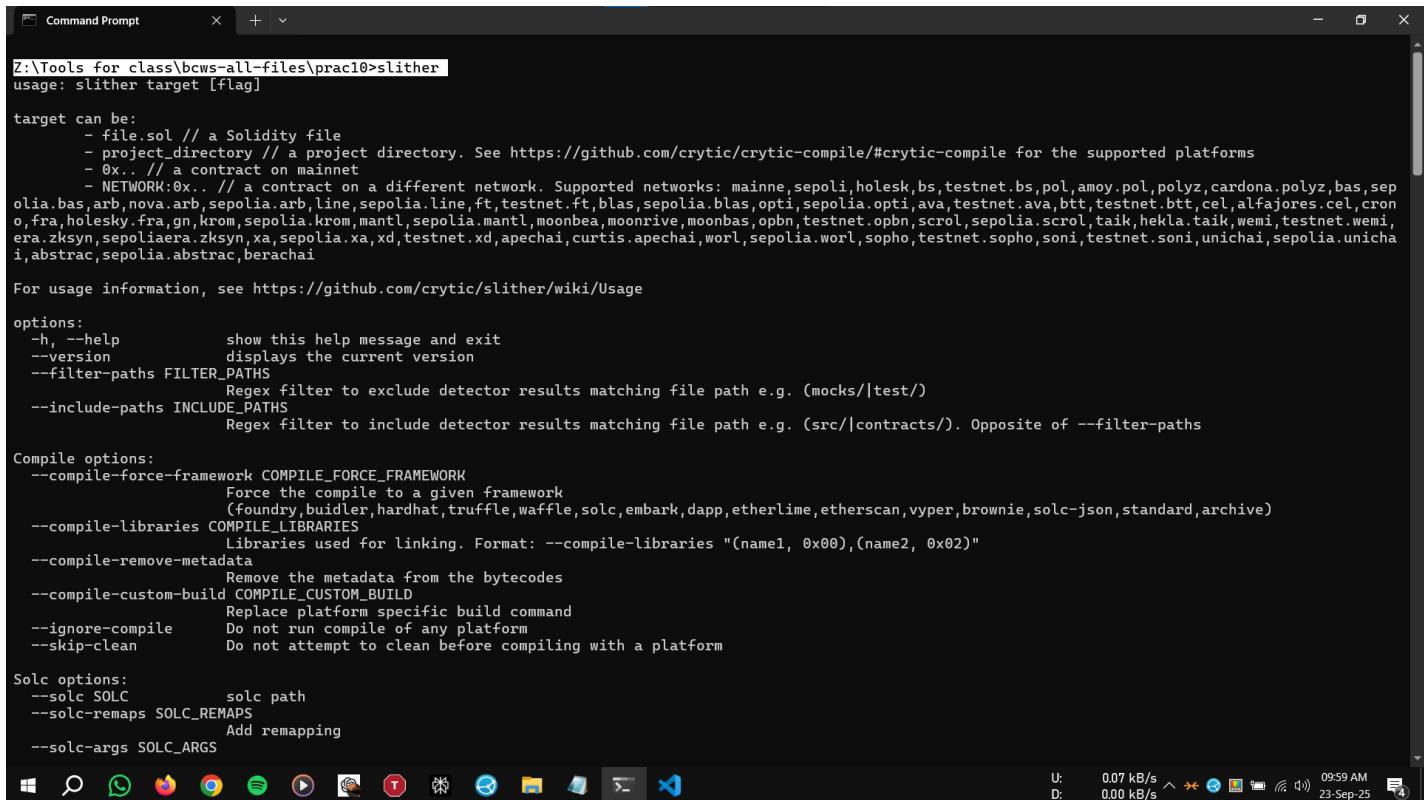
```
Z:\Tools for class\bcws-all-files\prac10>pip install slither-analyzer
Defaulting to user installation because normal site-packages is not writeable
Collecting slither-analyzer
  Downloading slither_analyzer-0.11.3-py3-none-any.whl.metadata (35 kB)
Requirement already satisfied: packaging in c:\users\faltu\appdata\roaming\python\python313\site-packages (from slither-analyzer) (24.2)
Collecting prettytable>=3.10.2 (from slither-analyzer)
  Downloading prettytable-3.10.2-py3-none-any.whl.metadata (33 kB)
Collecting pycryptodome>=3.4.6 (from slither-analyzer)
  Downloading pycryptodome-3.23.0-cp37abi3-win_amd64.whl.metadata (3.5 kB)
Collecting crytic-compile<0.4.0,>=0.3.9 (from slither-analyzer)
  Downloading crytic_compile-0.3.10-py3-none-any.whl.metadata (3.8 kB)
Collecting web3<8,>=7.10 (from slither-analyzer)
  Downloading web3-7.13.0-py3-none-any.whl.metadata (5.6 kB)
Collecting eth-abi>=5.0.1 (from slither-analyzer)
  Downloading eth_abi-5.2.0-py3-none-any.whl.metadata (3.8 kB)
Collecting eth-typing>=5.0.0 (from slither-analyzer)
  Downloading eth_typer-5.2.1-py3-none-any.whl.metadata (3.2 kB)
Collecting eth-utils>=5.0.0 (from slither-analyzer)
  Downloading eth_utils-5.3.1-py3-none-any.whl.metadata (5.7 kB)
Collecting cbor2 (from crytic-compile<0.4.0,>=0.3.9->slither-analyzer)
  Downloading cbor2-5.7.0-cp313-cp313-win_amd64.whl.metadata (5.6 kB)
Collecting solc-select>=1.0.4 (from crytic-compile<0.4.0,>=0.3.9->slither-analyzer)
  Downloading solc_select-1.1.0-py3-none-any.whl.metadata (46 kB)
Collecting parsimonious<0.11.0,>=0.10.0 (from eth-abi>=5.0.1->slither-analyzer)
  Downloading parsimonious-0.10.0-py3-none-any.whl.metadata (25 kB)
Requirement already satisfied: typing_extensions>=4.5.0 in c:\users\faltu\appdata\roaming\python\python313\site-packages (from eth-typing>=5.0.0->slither-analyzer) (4.14.1)
Collecting eth-hash>=0.3.1 (from eth-utils>=5.0.0->slither-analyzer)
  Downloading eth_hash-0.7.1-py3-none-any.whl.metadata (4.2 kB)
Collecting cytoolz>=0.10.1 (from eth-utils>=5.0.0->slither-analyzer)
  Downloading cytoolz-1.0.1-cp313-cp313-win_amd64.whl.metadata (4.7 kB)
Collecting pydantic<3,>=2.0.0 (from eth-utils>=5.0.0->slither-analyzer)
  Downloading pydantic-2.11.9-py3-none-any.whl.metadata (68 kB)
Requirement already satisfied: wcwidth in c:\users\faltu\appdata\roaming\python\python313\site-packages (from prettytable>=3.10.2->slither-analyzer) (0.2.13)
Collecting eth-account>=0.13.6 (from web3<8,>=7.10->slither-analyzer)
  Downloading eth_account-0.13.7-py3-none-any.whl.metadata (3.7 kB)
Collecting hexbytes>=1.2.0 (from web3<8,>=7.10->slither-analyzer)
  Downloading hexbytes-1.3.1-py3-none-any.whl.metadata (3.3 kB)
Collecting aiohttp>=3.7.4.post0 (from web3<8,>=7.10->slither-analyzer)
  Downloading aiohttp-3.12.15-cp313-cp313-win_amd64.whl.metadata (7.9 kB)
```

Step4: Open environment variable & add the Scripts folder to PATH.



Step5: Open cmd and run the command:

slither



```
Z:\Tools for class\bcws-all-files\prac10>slither
usage: slither target [flag]

target can be:
  - file.sol // a Solidity file
  - project_directory // a project directory. See https://github.com/crytic/crytic-compile/#crytic-compile for the supported platforms
  - 0x.. // a contract on mainnet
  - NETWORK:0x.. // a contract on a different network. Supported networks: mainne,sepolia,holesk,bs,testnet.bs,pol,amoy.pol,polyz,cardona.polyz,bas,sep
olia.bas,arb,nova.arb,sepolia.arb,line,sepolia.line,ft,testnet.ft,blas,sepolia.blas,opti,sepolia.opti,ava,testnet.ava,btt,testnet.btt,cel,alfajores.cel,cron
o,fra,holesky.fra,gn,krom,sepolia.krom,mantl,sepolia.mantl,moonbea,moonrive,moonbas,opbn,testnet.opbn,scrol,sepolia.scrol,taike,hekla.taike,wemi,testnet.wemi,
era,zksyn,sepoliaera.zksyn,xa,sepolia.xa,xd,testnet.xd,apechhai,curtis.apechhai,worf,sepolia.worf,sopho,testnet.sopho,soni,testnet.soni,unichai,sepolia.unicha
i,abstrac,sepolia.abstrac,berachai

For usage information, see https://github.com/crytic/slither/wiki/Usage

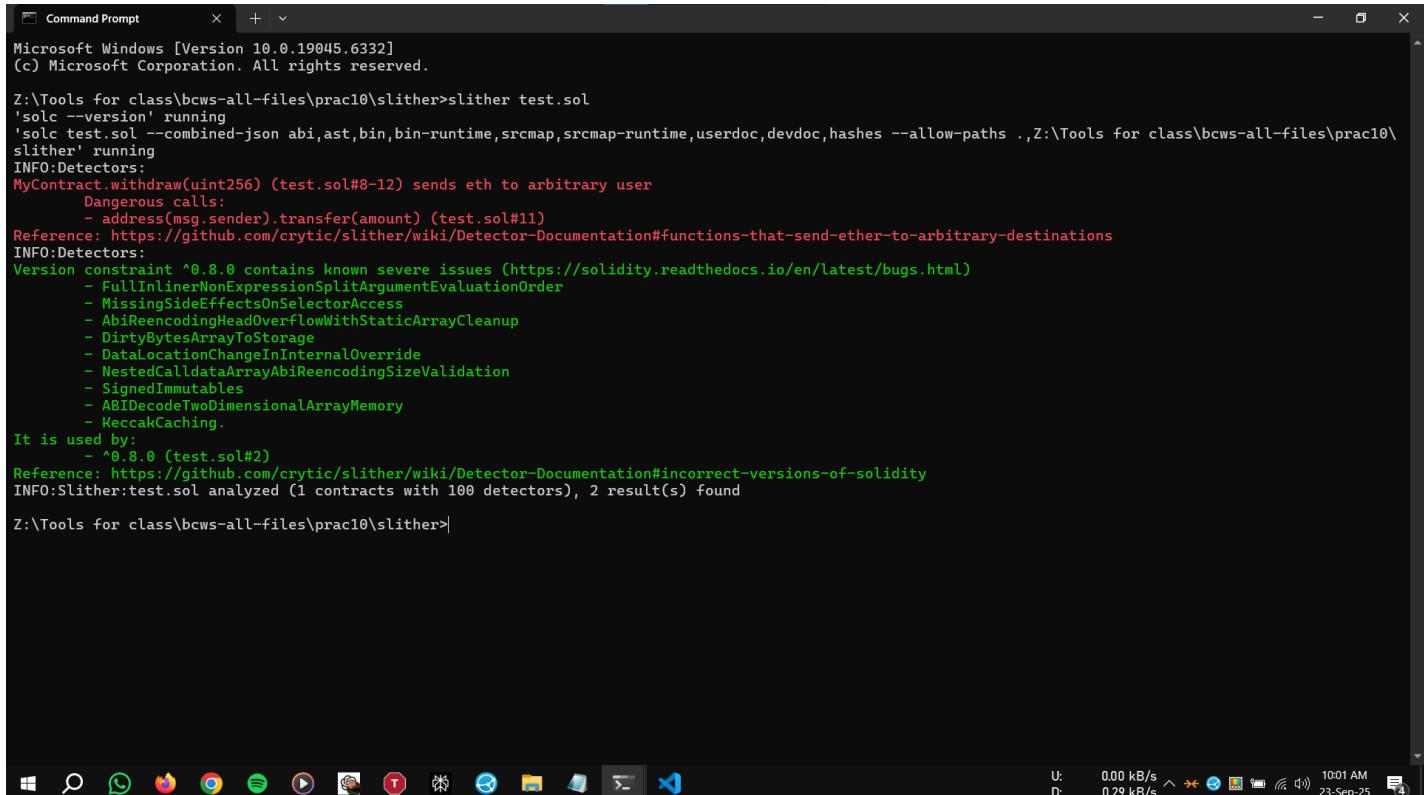
options:
  -h, --help           show this help message and exit
  --version          displays the current version
  --filter-paths FILTER_PATHS
    Regex filter to exclude detector results matching file path e.g. (mocks/|test/)
  --include-paths INCLUDE_PATHS
    Regex filter to include detector results matching file path e.g. (src/|contracts/). Opposite of --filter-paths

Compile options:
  --compile-force-framework COMPILE_FORCE_FRAMEWORK
    Force the compile to a given framework
    (foundry, builder, hardhat, truffle, solc, embark, dapp, etherlime, etherscan, vyper, brownie, solc-json, standard, archive)
  --compile-libraries COMPILE_LIBRARIES
    Libraries used for linking. Format: --compile-libraries "(name1, 0x00),(name2, 0x02)"
  --compile-remove-metadata
    Remove the metadata from the bytecodes
  --compile-custom-build COMPILE_CUSTOM_BUILD
    Replace platform specific build command
  --ignore-compile   Do not run compile of any platform
  --skip-clean       Do not attempt to clean before compiling with a platform

Solc options:
  --solc SOLC          solc path
  --solc-remaps SOLC_REMAPS
    Add remapping
  --solc-args SOLC_ARGS
```

Step6: Run the command given below to detect the vulnerabilities in the contract and also give the detail info about the vulnerabilities:

slither test.sol



```
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

Z:\Tools for class\bcws-all-files\prac10>slither test.sol
'solc --version' running
'solc test.sol --combined-json abi,ast,bin,bin-runtime,srcmap,srcmap-runtime,userdoc,devdoc,hashes --allow-paths ..,Z:\Tools for class\bcws-all-files\prac10\slither' running
INFO:Detectors:
MyContract.withdraw(uint256) (test.sol#8-12) sends eth to arbitrary user
  Dangerous calls:
    - address(msg.sender).transfer(amount) (test.sol#11)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Version constraint ^0.8.0 contains known severe issues (https://solidity.readthedocs.io/en/latest/bugs.html)
  - FullInlinerNonExpressionSplitArgumentEvaluationOrder
  - MissingSideEffectsOnSelectorAccess
  - AbiReencodingHeadOverflowWithStaticArrayCleanup
  - DirtyBytesArrayToStorage
  - DataLocationChangeInInternalOverride
  - NestedCallDataAbiReencodingSizeValidation
  - SignedImmutables
  - ABIDecodeTwoDimensionalArrayMemory
  - KeccakCaching.
It is used by:
  - ^0.8.0 (test.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Slither:test.sol analyzed (1 contracts with 100 detectors), 2 result(s) found

Z:\Tools for class\bcws-all-files\prac10>
```