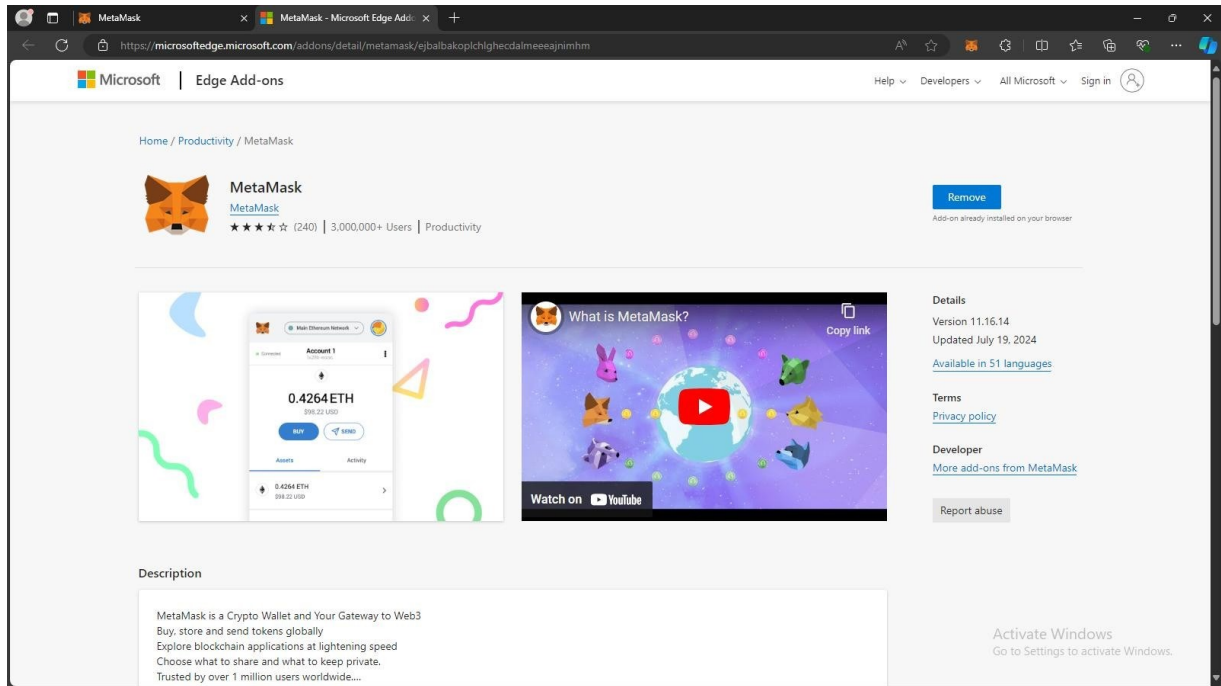


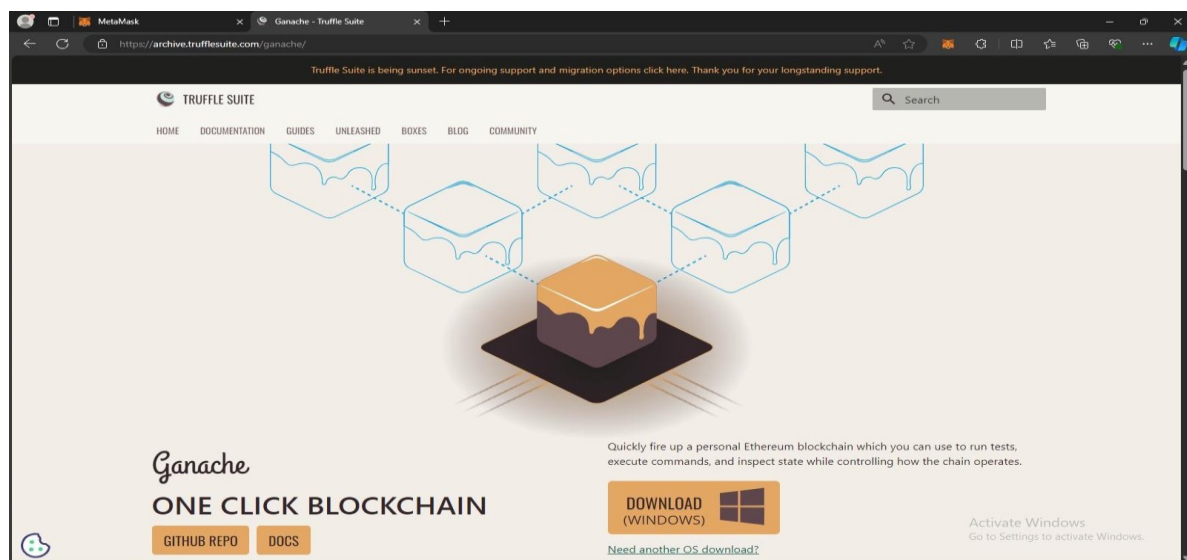
Practical 11: Creating a simple DApp for performing basic mathematical operations on two numbers.

Part 1: Setting up MetaMask and Ganache.

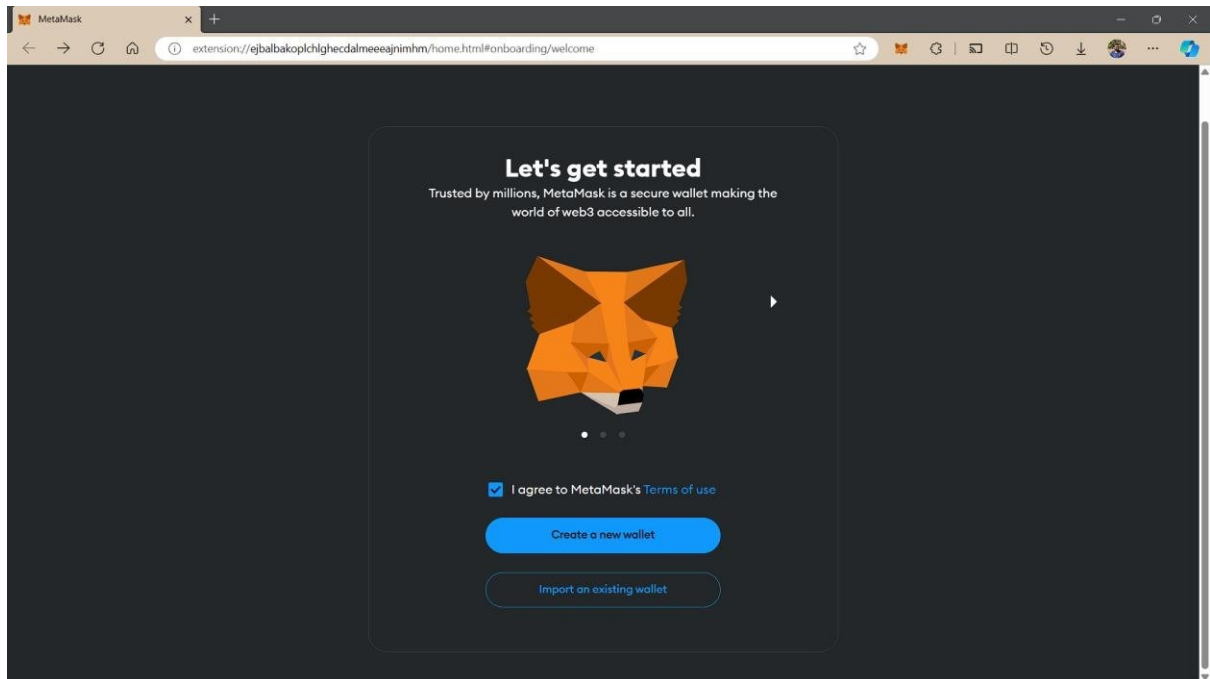
1. Install MetaMask Extension from [here](#).



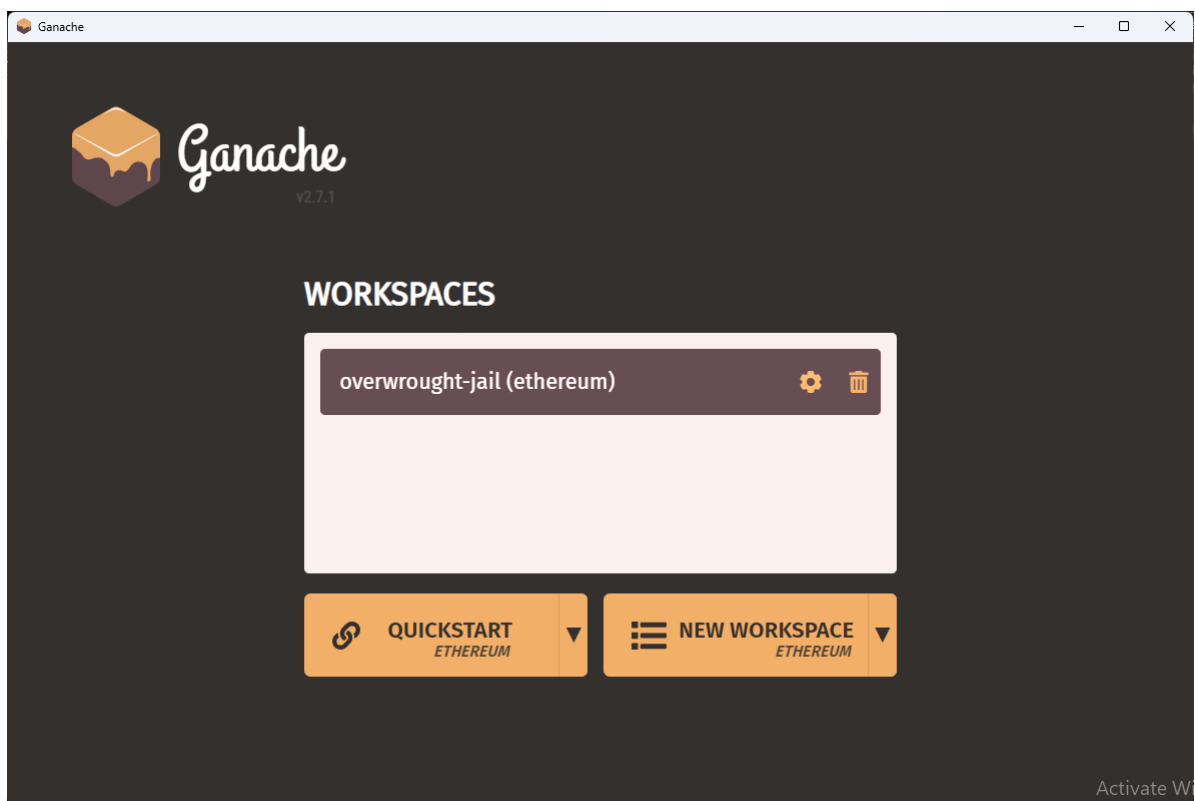
2. Install Ganache from [here](#).



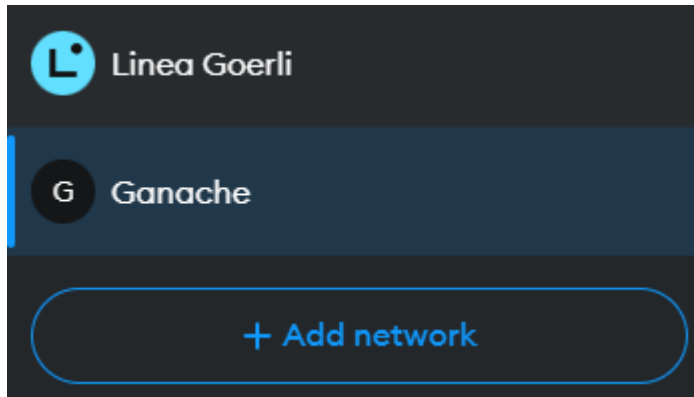
3. Create an Account on MetaMask or use an existing account.



4. Start a new workspace in Ganache.



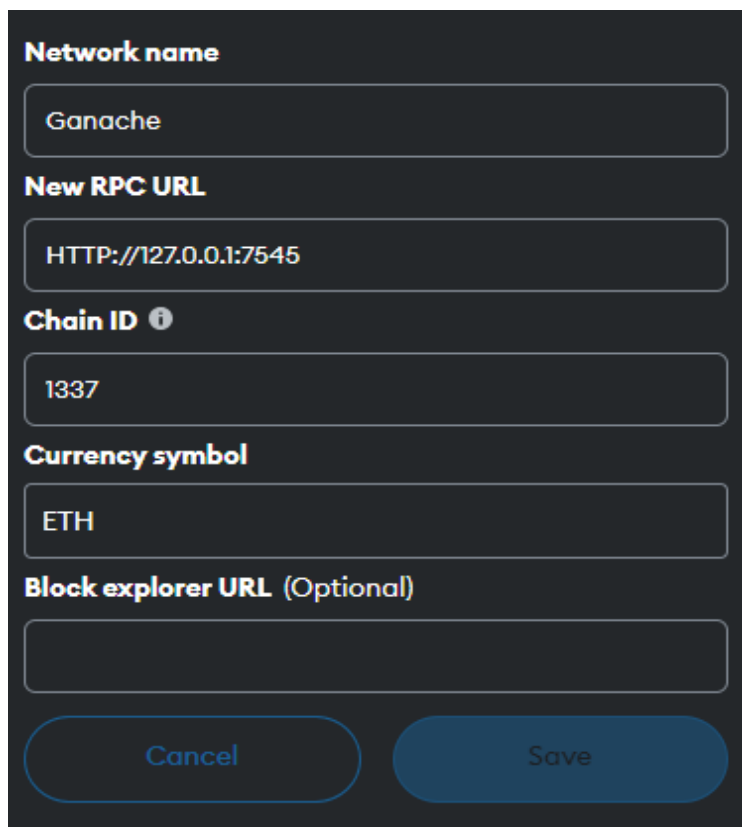
5. Go to MetaMask. Click on the Network Name → Add Network → Add Network Manually.



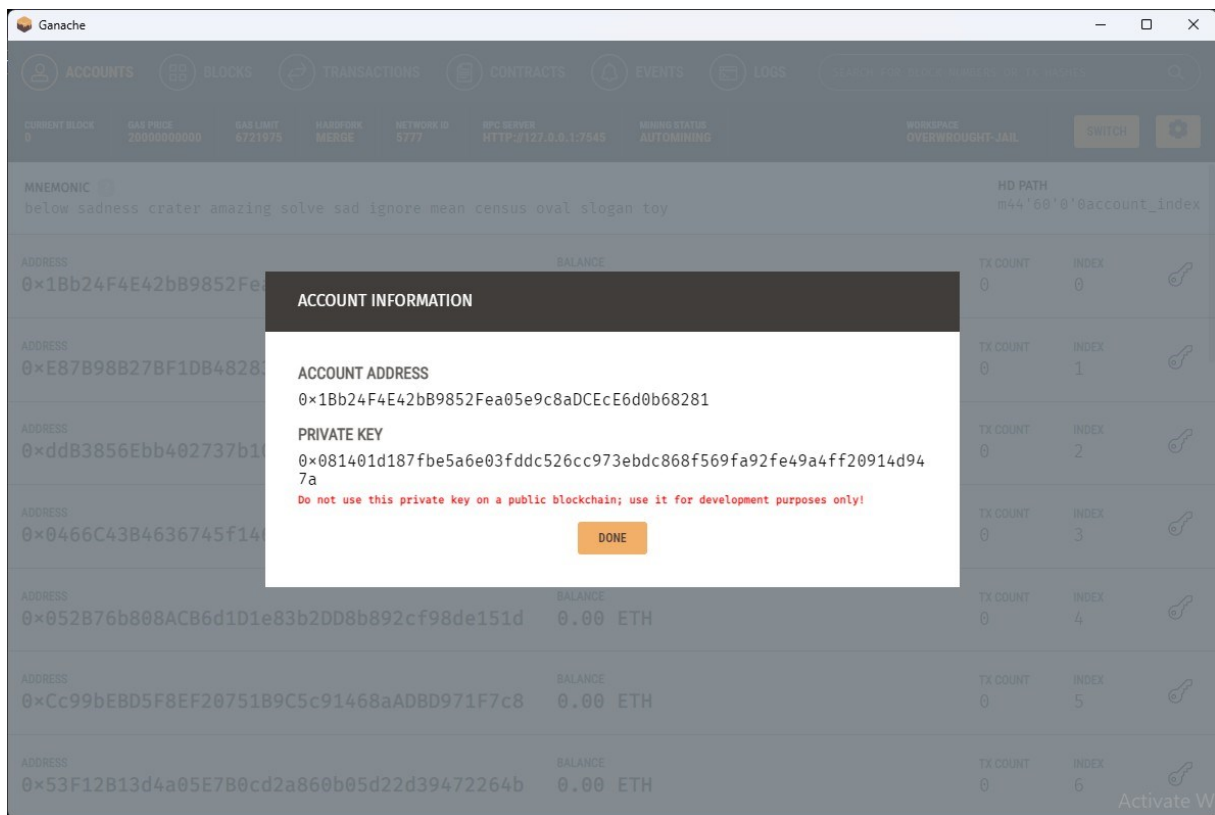
6. Give the Network a name of your choice.

7. Copy the RPC Server URL from Ganache GUI and paste it in the given box.
The default Chain ID is 1337.

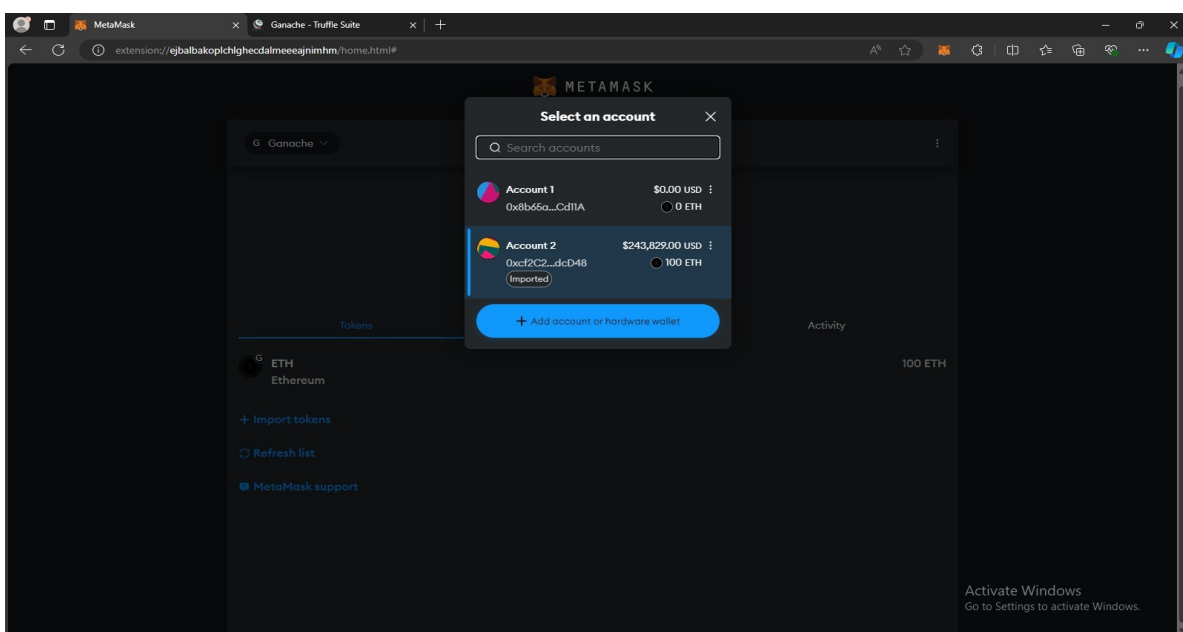
8. Give the currency symbol as ETH. Save the network.

A screenshot of the 'Add Network' form in the MetaMask application. The form is displayed on a dark background with white text. It contains several input fields: 'Network name' with the value 'Ganache', 'New RPC URL' with the value 'HTTP://127.0.0.1:7545', 'Chain ID' with the value '1337', 'Currency symbol' with the value 'ETH', and 'Block explorer URL (Optional)' which is currently empty. At the bottom of the form, there are two buttons: 'Cancel' and 'Save'. The 'Save' button is highlighted in a darker blue, indicating it is the primary action.

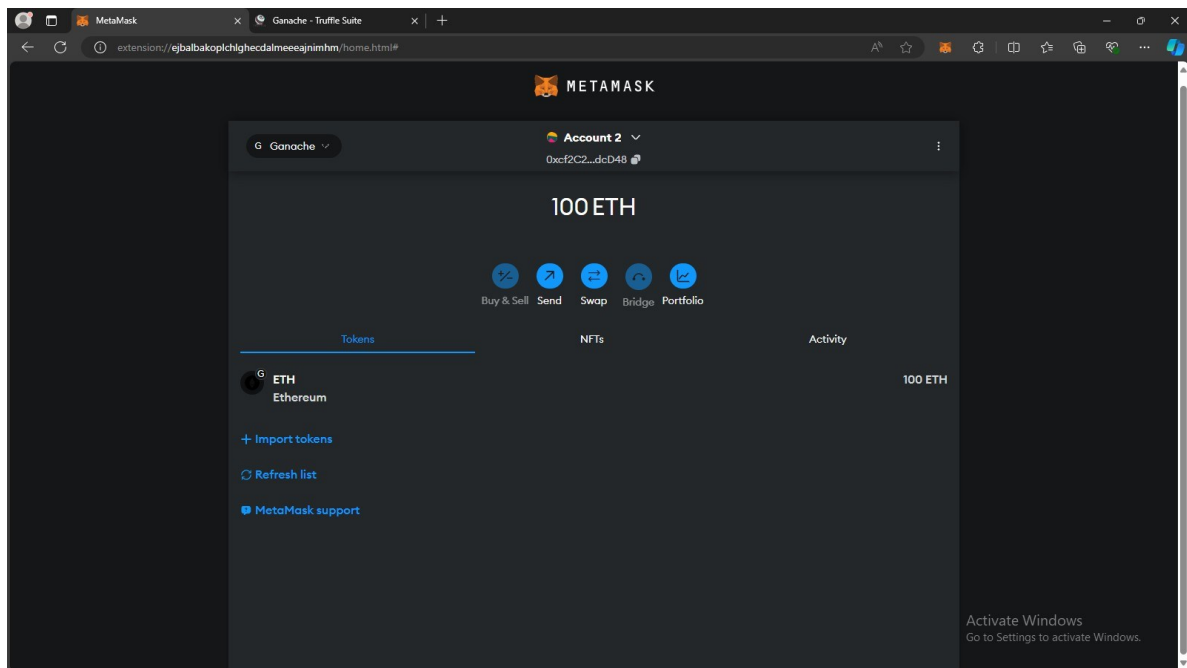
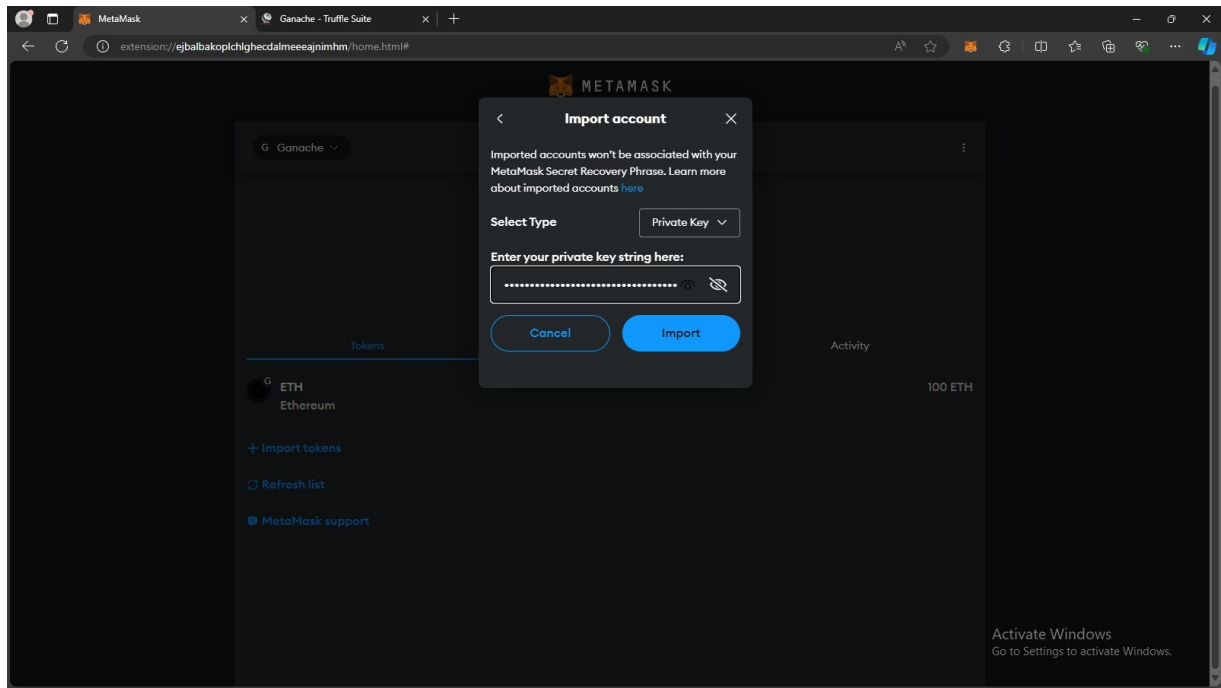
9. Go to Ganache and Click on the key symbol. Copy the private key.



10. Go to MetaMask and click on Account Name. Select Add Account or Hardware Wallet → Import Account and paste the private key in the given box.



Name : Pratik Tiruwa
Seat No : 31031523035



Part 2: Setting up VS Code.

Prerequisites:

a. Check if node and npm are installed with the following

commands `node -v` & `npm -v`

b. Install truffle, ganache & lite-server using the following command

`npm install -g truffle` / `npm install -g ganache` / `npm install lite-server --dev`

c. Ensure Ganache is running in the background

1. Open Terminal in VS Code and initialise Truffle with the following command

`truffle init`

```
PS D:\DApps> truffle init

Starting init...
=====

> Copying project files to D:\DApps

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
```

2. Create a new contract in the contracts folder. And write a smart contract for adding two number.

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

pragma solidity 0.8.19;

contract Addition {
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }
}
```

3. Create a new folder frontend and make `index.html` and `app.js` files inside.

`index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DApp-1</title>
</head>
<body>
  <h1>Blockchain Addition DApp</h1>
  <input type="number" id="num1" placeholder="Enter first number">
  <input type="number" id="num2" placeholder="Enter second number">
  <button onclick="addNumber()">Add Numbers</button>
  <h3>Result: <span id="result"></span></h3>

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

`app.js`

```
const contractAddress = ""; // Replace with your deployed contract address
const contractABI = []; // Use ABI from compiled contract

let web3;
let contract;

window.addEventListener("load", async () => {
  if (window.ethereum) {
    web3 = new Web3(window.ethereum);
    await window.ethereum.enable();
  } else {
    console.log("MetaMask not detected. Please install MetaMask.");
  }
});
```

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

    async function addNumber() {
        const num1 = document.getElementById("num1").value;
        const num2 = document.getElementById("num2").value;
        const accounts = await web3.eth.getAccounts();
        console.log(num1);
        console.log(num2);
        contract.methods
            .add(num1, num2)
            .call({ from: accounts[0] })
            .then((result) => {
                console.log(result);
                document.getElementById("result").innerText = `${result}`;
            });
    }
}
```

4. Create `1_deploy.js` in the migrations folder.

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

module.exports = async function (deployer) {
    await deployer.deploy(Addition);
    const instance = await Addition.deployed();
    console.log("Addition deployed at:", instance.address);
};
```

5. Create `test.js` in the test folder to verify the contracts before deploying it.

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

contract("Addition", () => {
    it("should add two numbers correctly", async () => {
        const addition = await Addition.deployed();
        console.log("Contract Address: ", addition.address);
        const result = await addition.add(5, 3);
    });
});
```



```
    assert.equal(result.toNumber(), 8, "Addition of 5 and 3 should be 8");  
  });  
});
```

6. In the source directory create a new file `bs-config.json` and set the base directory as frontend.

```
{  
  "server": {  
    "baseDir": ["/frontend"]  
  }  
}
```

7. Make sure about the following things

a. In the `truffle-config.js` uncomment your network details. And ensure the `port` and `network_id` match with the `RPC Server` which can be found in Ganache GUI

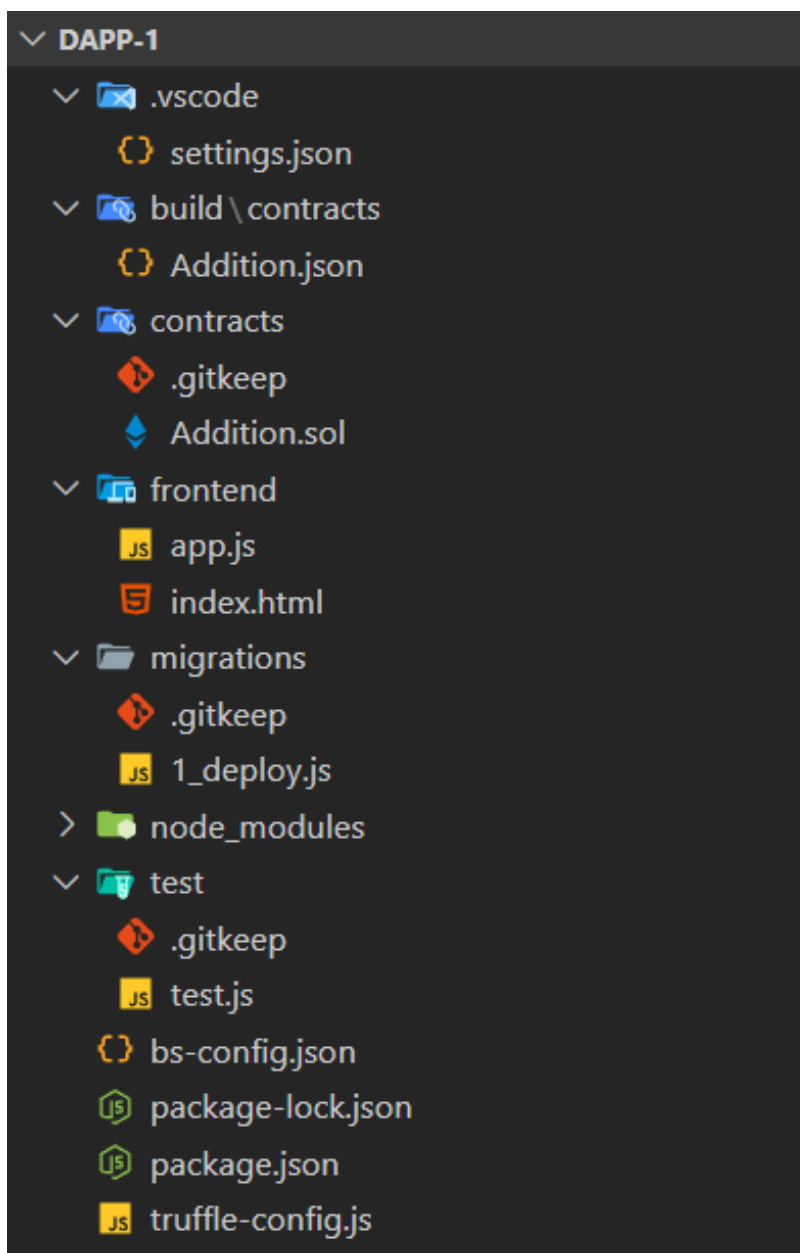
b. Ensure that solidity compiler version is set to 0.8.19 in the same file.

```
module.exports = {  
  networks: {  
    development: {  
      host: "127.0.0.1",  
      port: 7545,  
      network_id: "5777",  
    },  
  },  
  
  // Configure your compilers  
  compilers: {  
    solc: {  
      version: "0.8.19"  
    }  
  }  
};
```

a. Ensure necessary dependencies are mentioned in the `package.json`.

```
{  
  "dependencies": {  
    "lite-server": "^2.6.1"  
  },  
  "scripts": {  
    "start": "lite-server"  
  }  
}
```

b. The final directory structure should look like this



Part 3: Running the DApp.

1. In a new terminal set directory to source and run `truffle compile` command.

```
PS D:\Notes\PG\PG Sem 3\Practicals\Blockchain\DApp-1> truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\Addition.sol
> Compiling .\contracts\Addition.sol
> Artifacts written to D:\Notes\PG\PG Sem 3\Practicals\Blockchain\DApp-1\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang
```

2. Go to build → contracts → Addition.json. Look for `abi` and Copy the complete array. Paste it in the `contractABI` constant inside `app.js`.

```
const contractABI = [{
  inputs: [
    {
      internalType: "uint256",
      name: "a",
      type: "uint256",
    },
    {
      internalType: "uint256",
      name: "b",
      type: "uint256",
    },
  ],
  name: "add",
  outputs: [
    {
      internalType: "uint256",
      name: "",
      type: "uint256",
    },
  ],
  stateMutability: "pure",
  type: "function",
},
];
```

3. Next run `truffle migrate`. Make note of the Contract Address displayed in the terminal.

```
Compiling your contracts...
=====
> Compiling .\contracts\Addition.sol
> Artifacts written to D:\Notes\PG\PG Sem 3\Practicals\Blockchain\DApp-1\build\contracts
> Compiled successfully using:
  - solc: 0.8.19+commit.7dd6d404.Emscripten.clang

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

1 deploy.js
=====

Replacing 'Addition'
-----
> transaction hash: 0x791e3ab88d05b3012242b865bbd1105d39a645bcffaa0857f2c58ba562c22073
> Blocks: 0        Seconds: 0
> contract address: 0x9FC99312430F79737561207e8d26a2B92D3f280B
> block number:    24
> block timestamp: 1728397885
> account:         0xfFc0BF73A62a7A6b197fAB896164944A297B35
> balance:         99.99234039414607952
> gas used:        146899 (0x23dd3)
> gas price:       2.545027085 gwei
> value sent:      0 ETH
> total cost:      0.000373861933759415 ETH

Addition deployed at: 0x9FC99312430F79737561207e8d26a2B92D3f280B
> Saving artifacts
-----
> Total cost:      0.000373861933759415 ETH

Summary
=====
> Total deployments: 1
> Final cost:       0.000373861933759415 ETH
```

4. Copy the contract address and paste it in the `contractAddress` constant in the `app.js` file.

```
const contractAddress = "0xf3539bF7942055a8944EB7048A15450c05b1815A";
```

5. Run `truffle test` to ensure our contract is correct.

```
Using network 'development'.

Compiling your contracts...
=====
> Compiling .\contracts\Addition.sol
> Artifacts written to C:\Users\Admin\AppData\Local\Temp\test--19664-8nJHijHoKW2f
> Compiled successfully using:
  - solc: 0.8.19+commit.7dd6d404.Emscripten.clang
Addition deployed at: 0xb8D72a1caE30c216B0dE94f99f82b219A6f25A3c

Contract: Addition
Contract Address: 0xb8D72a1caE30c216B0dE94f99f82b219A6f25A3c
  ✓ should add two numbers correctly

1 passing (66ms)
```

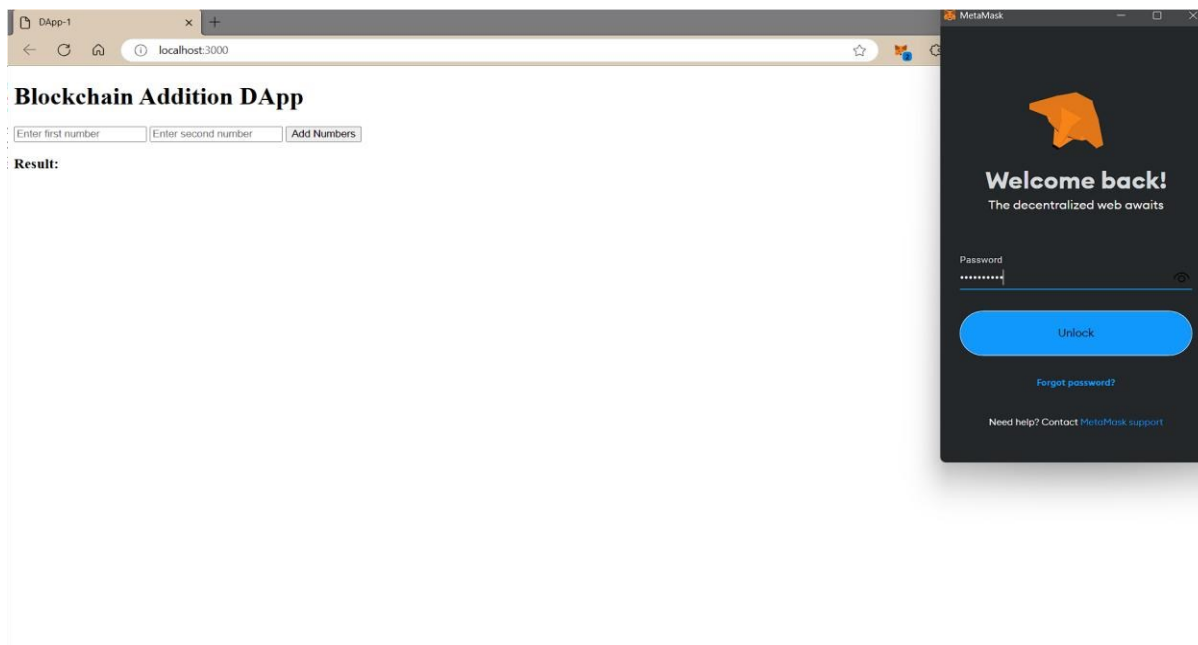
6. Run `npm start` if everything is correct.

```
PS D:\Notes\PG\PG Sem 3\Practicals\Blockchain\DApp-1> npm start

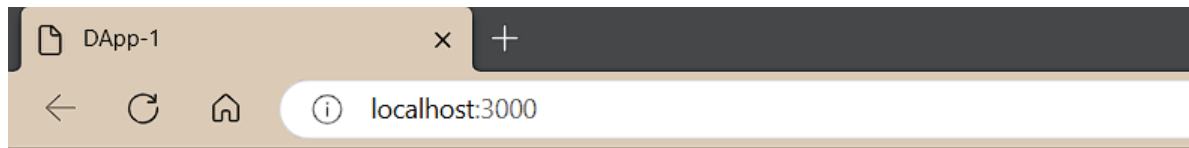
> start
> lite-server

** browser-sync config **
{
  injectChanges: false,
  files: [ './**/*.html,css,js' ],
  watchOptions: { ignored: 'node_modules' },
  server: {
    baseDir: [ './frontend' ],
    middleware: [ [Function (anonymous)], [Function (anonymous)] ]
  }
}
[Browsersync] Access URLs:
-----
    Local: http://localhost:3000
  External: http://192.168.0.224:3000
-----
    UI: http://localhost:3001
  UI External: http://localhost:3001
-----
[Browsersync] Serving files from: ./frontend
[Browsersync] Watching files...
24.10.08 20:12:58 200 GET /index.html
24.10.08 20:12:58 200 GET /app.js
24.10.08 20:12:58 404 GET /favicon.ico
```

7. Sign in to MetaMask and grant the required access.



8. Give the input and click on Add Numbers. The result should be displayed.



Blockchain Addition DApp

45 98 Add Numbers

Result: 143

Part 4: Modify the DApp to integrate subtraction, multiplication & division operations.

1. Make changes in the smart contract (Operations.sol → I have renamed thefile)

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

pragma solidity 0.8.19;

contract Operations {
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }

    function sub(uint256 a, uint256 b) public pure returns (uint256) {
        return a - b;
    }

    function mul(uint256 a, uint256 b) public pure returns (uint256) {
        return a * b;
    }

    function div(uint256 a, uint256 b) public pure returns (uint256) {
        return a / b;
    }
}
```

2. Modify index.html to accommodate other buttons and onClick functions.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>DApp-1</title>
  </head>
  <body>
    <h1>Blockchain Addition DApp</h1>
```

```
Number 1:
<input type="number" id="num1" placeholder="Enter first number" />
<br /><br />
Number 2:
<input type="number" id="num2" placeholder="Enter second number" />
<br />
<h2>Choose Operation:</h2>
<button onclick="addNumber()">Add</button>
<button onclick="subNumber()">Sub</button>
<button onclick="mulNumber()">Mul</button>
<button onclick="divNumber()">Div</button>
<h2>Result:</h2>
<h3>Addition: <span id="resultA"></span></h3>
<h3>Subtraction: <span id="resultS"></span></h3>
<h3>Multiplication: <span id="resultM"></span></h3>
<h3>Division: <span id="resultD"></span></h3>
<script
src="https://cdn.jsdelivr.net/npm/web3@latest/dist/web3.min.js"></script>
<script src="app.js"></script>
</body>
</html>
```

3. Similarly modify app.js.

```
const contractAddress = ""; // Replace with your deployed contract address
const contractABI = []; // Use ABI from compiled contract

let web3;
let contract;

window.addEventListener("load", async () => {
  if (window.ethereum) {
    web3 = new Web3(window.ethereum);
    await window.ethereum.enable();
  } else {
    console.log("MetaMask not detected. Please install MetaMask.");
  }
})
```



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

async function addNumber() {
  const num1 = document.getElementById("num1").value;
  const num2 = document.getElementById("num2").value;
  const accounts = await web3.eth.getAccounts();
  console.log(num1);
  console.log(num2);
  contract.methods
    .add(num1, num2)
    .call({ from: accounts[0] })
    .then((result) => {
      console.log(result);
      document.getElementById("resultA").innerText = `${result}`;
    });
}

async function subNumber() {
  const num1 = document.getElementById("num1").value;
  const num2 = document.getElementById("num2").value;
  const accounts = await web3.eth.getAccounts();
  console.log(num1);
  console.log(num2);
  contract.methods
    .sub(num1, num2)
    .call({ from: accounts[0] })
    .then((result) => {
      console.log(result);
      document.getElementById("resultS").innerText = `${result}`;
    });
}

async function mulNumber() {
  const num1 = document.getElementById("num1").value; const num2 =
  document.getElementById("num2").value; const accounts = await web3.eth.getAccounts();
```

```
    console.log(num2);
    contract.methods
      .mul(num1, num2)
      .call({ from: accounts[0] })
      .then((result) => {
        console.log(result);
        document.getElementById("resultM").innerText = `${result}`;
      });
  }

  async function divNumber() {
    const num1 = document.getElementById("num1").value;
    const num2 = document.getElementById("num2").value;
    const accounts = await web3.eth.getAccounts();
    console.log(num1);
    console.log(num2);
    contract.methods
      .div(num1, num2)
      .call({ from: accounts[0] })
      .then((result) => {
        console.log(result);
        document.getElementById("resultD").innerText = `${result}`;
      });
  }
}
```

4. Modify 1_deploy.js (If you haven't renamed leave it as is)

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

module.exports = async function (deployer) {
  await deployer.deploy(Operations);
  const instance = await Operations.deployed();
  console.log("Operations deployed at:", instance.address);
};
```

5. Update test.js to include different test cases.

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

contract("Operations", () => {
  let operationsInstance;

  before(async () => {
    operationsInstance = await Operations.deployed();
  });

  it("should add two numbers correctly", async () => {
    console.log("Contract Address: ", operationsInstance.address);
    const resultA = await operationsInstance.add(5, 2);
    assert.equal(resultA.toNumber(), 7, "Addition of 5 and 2 should be 7");
  });

  it("should subtract two numbers correctly", async () => {
    console.log("Contract Address: ", operationsInstance.address);
    const resultS = await operationsInstance.sub(5, 2);
    assert.equal(resultS.toNumber(), 3, "Subtraction of 5 and 2 should be 3");
  });

  it("should multiply two numbers correctly", async () => {
    console.log("Contract Address: ", operationsInstance.address);
    const resultM = await operationsInstance.mul(5, 2);
    assert.equal(resultM.toNumber(), 10, "Multiplication of 5 and 2 should be 10");
  });

  it("should divide two numbers correctly", async () => {
    console.log("Contract Address: ", operationsInstance.address);
    const resultD = await operationsInstance.div(5, 5);
    assert.equal(resultD.toNumber(), 1, "Division of 5 and 5 should be 1");
  });
});
```

6. Run the DApp following the same steps in Part 3.

DApp-1

×

+

←

↺

🏠

ⓘ

localhost:3000

Blockchain Operations DApp

Number 1:

Number 2:

Choose Operation:

Add

Sub

Mul

Div

Result:

Addition: 522

Subtraction: 350

Multiplication: 37496

Division: 5