Experiment 6

Aim: To demonstrate Structures in Solidity.

Description:

A Structure is a user-defined data type that allows a programmer to create their own data type based on requirement and use case.

Along with many programming languages, Solidity also supports Structures with the struct keyword.

Syntax:

struct name {

datatype1 var1;

datatype2 var2;

datatype3 var3;

}

Map: Mapping in Solidity acts like a hash table or dictionary in any other language. These are used to store the data in the form of key-value pairs, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type. Mappings are mostly used to associate the unique Ethereum address with the associated value type.

Syntax:

mapping(key => value) <access specifier> <name>;

Program:

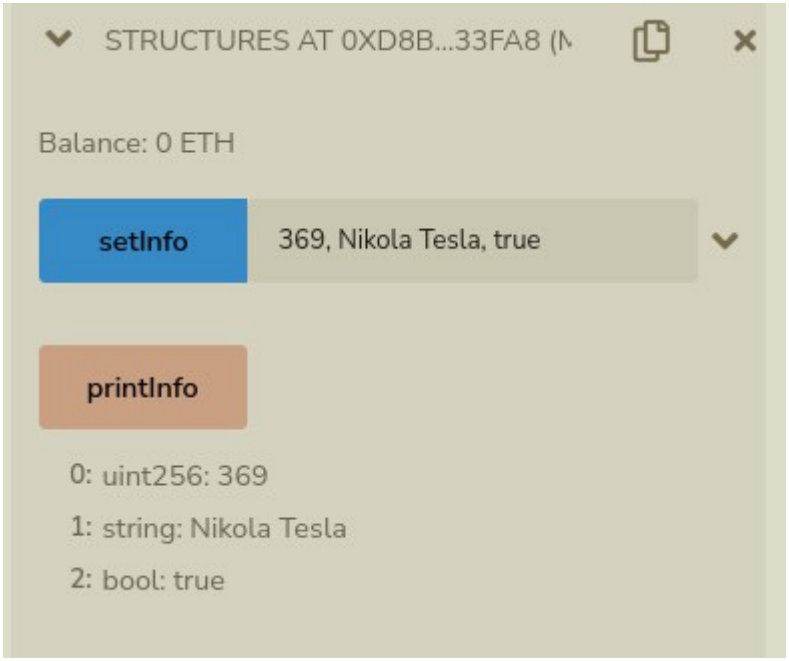
```
1 // SPDX-License-Identifier: MIT
2 pragma solidity > 0.6.0;
3
4 contract Structures {
5    struct StudentInfo {
6       uint roll_no;
7       string name;
8       bool has_chosen_elective;
9    }
10
```

```
    StudentInfo s1;
    function printInfo() public view returns (uint, string memory, bool) {
        return (s1.roll_no, s1.name, s1.has_chosen_elective);
    }

    function setInfo(uint roll_no, string memory name, bool has_chosen_elective)
public {
        s1.roll_no = roll_no;
        s1.name = name;
        s1.has_chosen_elective = has_chosen_elective;
    }
}
```

Mappings:

```
pragma solidity ^0.4.18;
 contract Maps {
   struct student {
      string name;
      string subject;
      uint8 marks;
    }
   mapping (address => student) result;
   address[] student_result;
    function adding_values() public {
       var student = result[0xDEE7796E89C82C36BAdd1475476f39D69FafE252];
       student.name = "John";
       student.subject = "Chemistry";
       student.marks = 88;
       student_result.push(0xDEE7796E89C82C36BAdd1475476f39D69FafE252) -1;
    }
    function get_student_result( ) view public returns (address[]) {
       return student_result;
    }
    function count_students( ) view public returns (uint) {
       return student_result.length;}
}

```

Output:


STRUCTURES AT 0XD8B...33FA8 (M
Balance: 0 ETH
setInfo
369, Nikola Tesla, true
printInfo
0: uint256: 369
1: string: Nikola Tesla
2: bool: true


adding_values
count_students
0: uint256: 1
get_student_result
0: address[]: 0xDEE7796E89C82C36BAdd1375076f39D69FafE252

Experiment 7

Aim: To demonstrate the Mathematical and Cryptographic Functions in Solidity.

Description:

Solidity provides the following functions built-in to the language:

sha256(): Returns the SHA256 hash value of given input.

keccak256(): Returns the SHA256 hash value of given input.

ripemd160(): Returns the SHA256 hash value of given input.

addmod(x, y, z): Returns the value of (x+y)%z.

mulmod(): Returns the value of (x*y)%z.

abi.encodePacked(): Packs the given data into bytes by following the ABI encoding rules.

Program:

```
1  // SPDX-License-Identifier: LGPLv3
2  pragma solidity > 0.6.0;
3  contract HashFunctions {
4     function getSHA256(string memory input) public pure returns (bytes32) {
5        return sha256(abi.encodePacked(input));
6     }
7     function getKeccak256(string memory input) public pure returns (bytes32) {
8        return keccak256(abi.encodePacked(input));
9     }
10    function getRipemd160(string memory input) public pure returns (bytes32) {
11       return ripemd160(abi.encodePacked(input));
12    }
13    function getAddmod(uint x, uint y, uint z) public pure returns (uint) {
14       return addmod(x, y, z);
15    }
16    function getMulmod(uint x, uint y, uint z) public pure returns (uint) {
17       return mulmod(x, y, z);
```

Experiment 8

Aim: To setup and install Hyperledger Fabric.

Description:

Hyperledger Fabric is a permissioned blockchain infrastructure, originally contributed by IBM and Digital Asset, providing a modular architecture with a delineation of roles between the nodes in the infrastructure, execution of Smart Contracts (called "chaincode" in Fabric) and configurable consensus and membership services.

Permissioned network: Access to the network is granted only to known participants who have been granted permission to access the network.

Modularity: Fabric is highly modular, which allows organizations to choose the components they need for their specific use case. This modularity also enables Fabric to be more flexible and adaptable to different industries.

Chain code: Chain code is the smart contract technology used in Hyperledger Fabric. Chain code is written in a supported programming language such as Go, Java, or Node.js.

Private transactions: Transactions on Hyperledger Fabric can be kept private between the transacting parties. This is useful in cases where sensitive information is being exchanged.

Scalability: Hyperledger Fabric can handle large numbers of transactions per second and can scale horizontally by adding more nodes to the network.

Consensus: Hyperledger Fabric uses a pluggable consensus mechanism, which means organizations can choose the consensus mechanism that best suits their needs.

Procedure and Output:

1) Install the prerequisite programs git and curl.

```
sudo apt install git curl
```

2) Install and set-up docker.

```
sudo apt install docker.io

sudo systemctl enable –now docker

sudo usermod -aG docker $USER
```

Restart the system.

Verify the install.

```
docker-compose --version
```

3) Installation of Fabric.

```
curl -sSL https://bit.ly/2ysbOFE | bash -s
```

The above command clones a sample deployment of fabric followed by an installation of fabric.

```
1  ~/code/blockchain/fabric/fabric-samples/test-network on  main  ❯
./network.sh
2  Using docker and docker-compose
3  Usage:
4    network.sh <Mode> [Flags]
5      Modes:
6        up - Bring up Fabric orderer and peer nodes. No channel is created
7        up createChannel - Bring up fabric network with one channel
8        createChannel - Create and join a channel after the network is created
9        deployCC - Deploy a chaincode to a channel (defaults to asset-transfer-basic)
10        down - Bring down the network
11
12      Flags:
13      Used with network.sh up, network.sh createChannel:
14      -ca <use CAs> -  Use Certificate Authorities to generate network crypto material
15      -c <channel name> - Name of channel to create (defaults to "mychannel")
16      -s <dbtype> - Peer state database to deploy: goleveldb (default) or couchdb
17      -r <max retry> - CLI times out after certain number of attempts (defaults to 5)
18      -d <delay> - CLI delays for a certain number of seconds (defaults to 3)
19      -verbose - Verbose mode
20
21      Used with network.sh deployCC
22      -c <channel name> - Name of channel to deploy chaincode to
23      -ccn <name> - Chaincode name.
24      -ccl <language> - Programming language of the chaincode to deploy: go, java,
javascript, typescript
```

4) Run the test network

```
cd fabric-samples/test-network

./network.sh up
```

```
 1  ~/code/blockchain/fabric/fabric-samples/test-network on  main ✓ ❯
./network.sh up
 2
 3  Using docker and docker-compose
 4  Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using
database 'leveldb'
 5  LOCAL_VERSION=v2.5.0
 6  DOCKER_IMAGE_VERSION=v2.5.0
 7
 8  Creating network "fabric_test" with the default driver
 9  Creating volume "net_orderer.example.com" with default driver
10  Creating volume "net_peer0.org1.example.com" with default driver
11  Creating volume "net_peer0.org2.example.com" with default driver
12  Creating peer0.org2.example.com ... done
13  Creating orderer.example.com    ... done
14  Creating peer0.org1.example.com ... done
15  Creating cli                    ... done
16  CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS
PORTS                                        NAMES
17  1667543b5634   hyperledger/fabric-tools:latest     "/bin/bash"         1 second ago
Up Less than a second                                    cli
18  b6b117c81c7f   hyperledger/fabric-peer:latest      "peer node start"   2 seconds ago
Up 1 second          0.0.0.0:7051->7051/tcp                  peer0.org1.example.com
19  703ead770e05   hyperledger/fabric-orderer:latest   "orderer"           2 seconds ago
Up Less than a second   0.0.0.0:7050->7050/tcp, 0.0.0.0:7053->7053/tcp
orderer.example.com
20  718d43f5f312   hyperledger/fabric-peer:latest      "peer node start"   2 seconds ago
Up 1 second          7051/tcp, 0.0.0.0:9051->9051/tcp        peer0.org2.example.com
```

This command will create a network with one orderer, one peer, and one CA (Certificate Authority).

Experiment 9

Aim: To demonstrate the Interplanetary File System (IPFS).

Description:

The InterPlanetary File System (IPFS) is a protocol, hypermedia and file sharing peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting IPFS hosts.

Procedure and Outputs:

1. Visit the IPFS website as a reference for installation instructions.

2. Install ipfs.

```
1  snap install ipfs
```

3. Initialize the ipfs installation.

```
1  ipfs init
```

~ ❯ ipfs init

```
generating ED25519 keypair...done
peer identity: 12D3KooWR29FfPJSaSyTjM1F3vydRxZLWZn92dMSf9Xm3jqvD1br
initializing IPFS node at /home/ksk/.ipfs
to get started, enter:

        ipfs cat
/ipfs/QmQPeNsJPyVWPFDVHb77w8G42Fvo15z4bG2X8D2GhfbSXc/readme
```

4. Start the daemon.

```
1  ipfs daemon
```

~ ❯ ipfs daemon

```
Initializing daemon...
Kubo version: 0.19.1-958e586ca
Repo version: 13
System version: amd64/linux
```

Golang version: go1.20.3

Computed default go-libp2p Resource Manager limits based on:

- 'Swarm.ResourceMgr.MaxMemory': "4.0 GB"

- 'Swarm.ResourceMgr.MaxFileDescriptors': 262144

Theses can be inspected with 'ipfs swarm resources'.

Swarm listening on /ip4/127.0.0.1/tcp/4001

Swarm listening on /ip4/127.0.0.1/udp/4001/quic

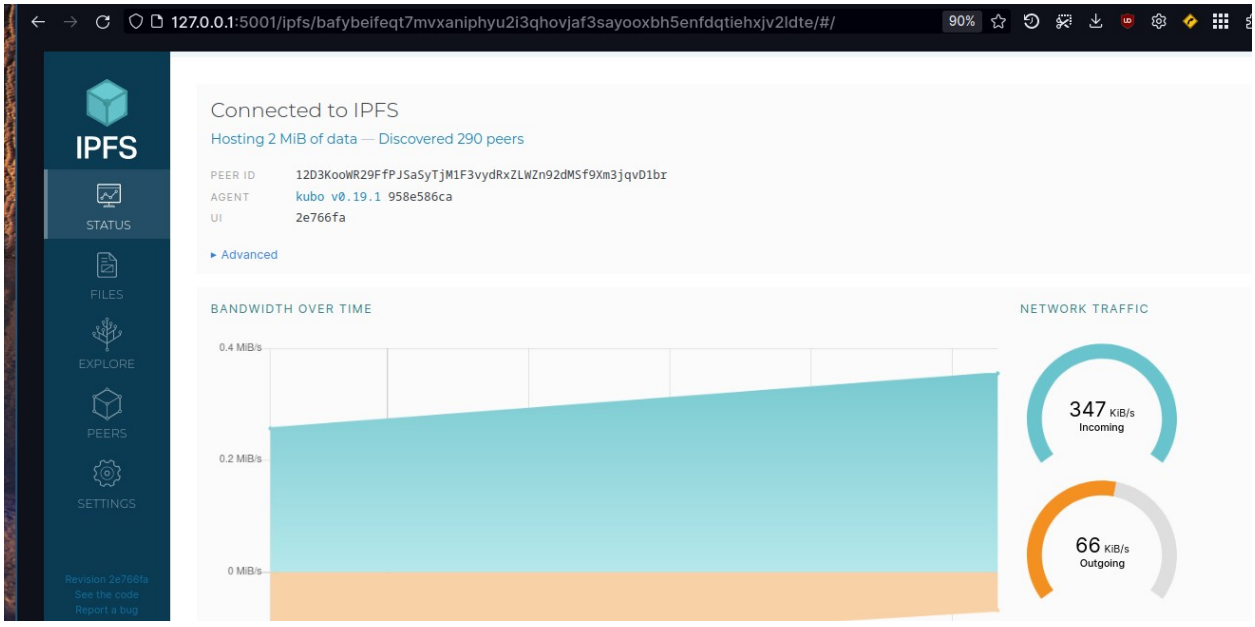
Swarm listening on /ip4/127.0.0.1/udp/4001/quic-v1

Swarm listening on /ip4/127.0.0.1/udp/4001/quic-v1/webtransport/certhash/uEiCVJkm5FstoXaoB6kbkZUvsdEXyDP29kNGcOGt02F9mxw/certhash/uEiApPWtL8aEYhMOaZI9gRSWZQQtsglZ7uSGcI7TOmMzS4Q

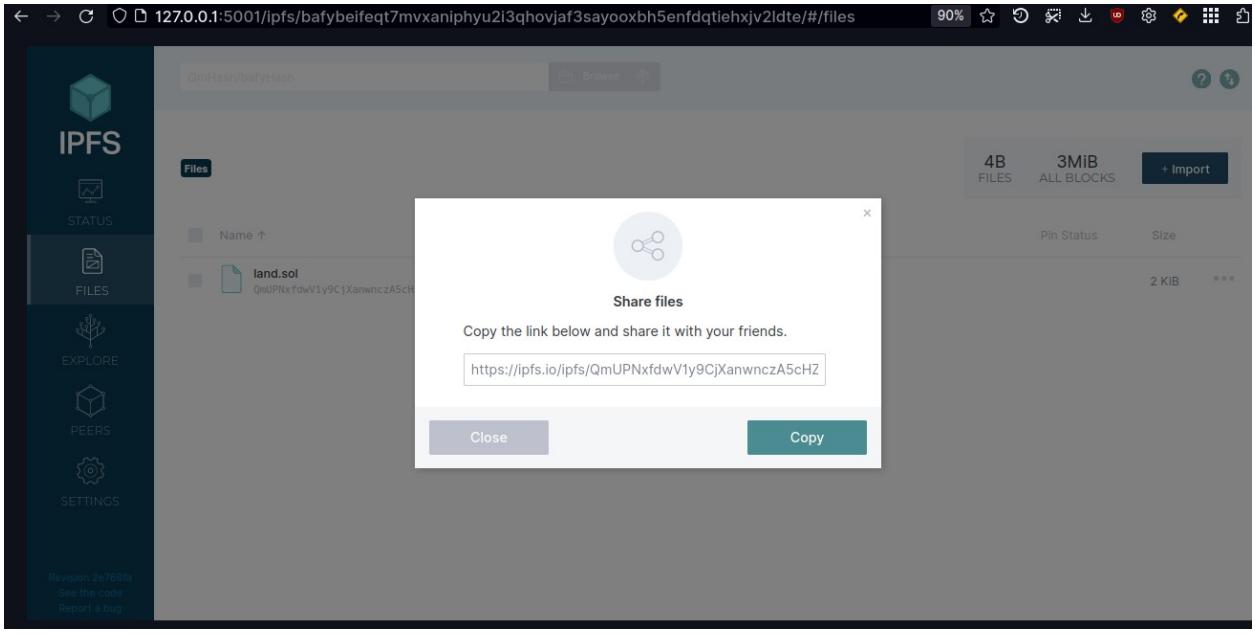
Swarm listening on /ip4/172.17.0.1/tcp/4001

5. Access the Web Interface by reading the “WebUI” line in the above output.

WebUI: http://127.0.0.1:5001/webui



7. Host your files by uploading them to the WebUI in the Files section.



A share link can be generated from the context menu to the right of the file.

Experiment 10 - Dapps

Aim: To explore Dapps and launch using Ethereum

Description:

A dApp (decentralized application) is a type of software application that runs on a decentralized network, such as a blockchain. Unlike traditional applications that run on centralized servers, dApps are designed to be decentralized, meaning they operate on a distributed network of computers or nodes, and the data is stored on a public ledger that is tamper-proof.

One of the key features of dApps is that they are open-source, meaning their source code is available for anyone to view, modify, or enhance. This enables anyone to contribute to the development of the dApp and ensure its integrity.

Another important feature of dApps is that they often use smart contracts, which are self-executing contracts with the terms of the agreement directly written into the code. This allows dApps to automate complex processes, such as payments or supply chain management, without requiring intermediaries or third parties.

One of the advantages of dApps is that they can be more transparent and secure than traditional applications, as the data is stored on a decentralized network that is difficult to hack or corrupt. Additionally, dApps can provide greater control and ownership to users, as they can interact with the application without needing to rely on a central authority.

However, there are also challenges to developing and using dApps, such as scalability issues and the complexity of programming on decentralized networks. Overall, dApps are an exciting development in the world of software applications, offering new opportunities for innovation and collaboration.

Installation Steps:

Installing a dApp (decentralized application) can vary depending on the specific dApp and the platform you are using. However, here are some general steps you can follow to install a dApp:

1. Choose your preferred platform: dApps can be installed on various platforms such as Ethereum, EOS, TRON, etc. Choose the platform that supports the dApp you want to install.

2. Install a compatible wallet: dApps require a wallet to store and manage cryptocurrencies. Choose a compatible wallet for the platform you have chosen. For example, for Ethereum, you can install Metamask, for EOS, you can install Scatter, and for TRON, you can install TronLink.
3. Fund your wallet: Most dApps require you to have some cryptocurrency to use them. Fund your wallet with the cryptocurrency required by the dApp you want to install.

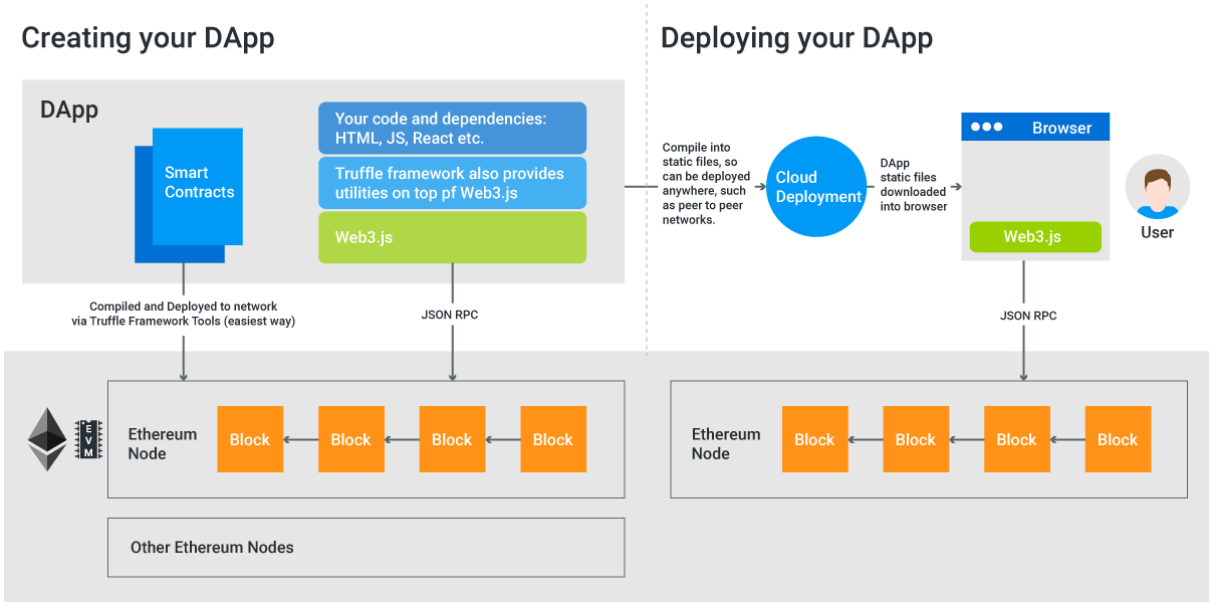
4. Locate the dApp: Find the dApp you want to install. You can search for dApps on the platform's website or through a dApp store.

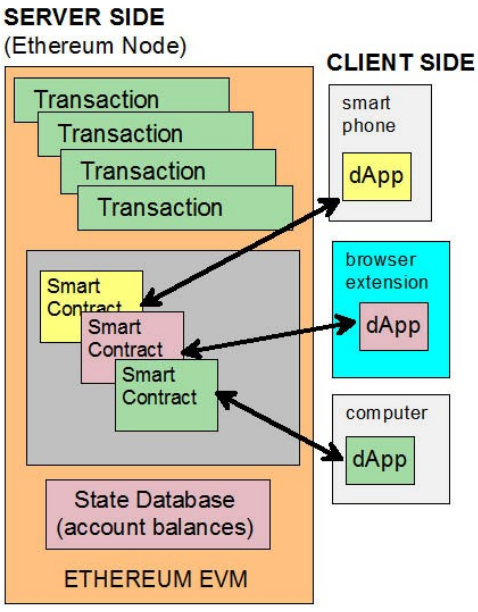
5. Install the dApp: Follow the instructions provided by the dApp to install it. In most cases, you will need to approve the installation using your wallet.

6. Use the dApp: Once the dApp is installed, you can use it as per its instructions. Some dApps may require you to connect your wallet to use them.

Note: Keep in mind that dApps are still in the early stages of development, and the installation process may vary depending on the specific dApp and platform. Always follow the instructions provided by the dApp to avoid any issues.

Screenshots:





Experiment 11 Case Study

Case Study: Blockchain in Supply Chain with Smart Contract

Background: Supply chain management is a complex process that involves multiple parties and transactions, making it susceptible to errors, fraud, and delays. The use of blockchain technology with smart contracts can help address these issues and improve the efficiency and transparency of the supply chain. In this case study, we will explore the application of blockchain in supply chain management with the use of smart contracts.

Problem: A major challenge in the supply chain is the lack of transparency and traceability, which makes it difficult to track products, identify the source of problems, and prevent fraud. Additionally, supply chain processes are often manual and time-consuming, leading to delays and increased costs.

Solution: Blockchain technology can provide a decentralized and secure platform for recording and tracking transactions in the supply chain. A smart contract is a self-executing contract that contains the rules and regulations of the transaction. By using blockchain with smart contracts, supply chain transactions can be automated and streamlined, increasing efficiency and reducing the potential for errors and fraud.

Case Study: A food manufacturing company, ABC Foods, is using blockchain with smart contracts to improve its supply chain management. The company sources raw materials from different suppliers and sells finished products to distributors and retailers.

The company implemented a blockchain-based platform that allows suppliers to upload data on the quality, quantity, and origin of the raw materials. The data is stored on the blockchain, providing a secure and transparent record of the transaction. Smart contracts are used to automate the process of verifying the data and releasing payment to the supplier.

Each product is assigned a unique identifier that is recorded on the blockchain. The smart contract is programmed to release payment to the logistics provider only after the delivery is confirmed by the distributor or retailer.

Benefits: By using blockchain with smart contracts, ABC Foods has improved the efficiency and transparency of its supply chain management. The benefits of using blockchain with smart contracts in the supply chain include:

1. Increased transparency: Blockchain provides a transparent and secure platform for recording transactions, allowing all parties to have access to the same information.
2. Reduced fraud: Smart contracts automate the verification process, reducing the potential for errors and fraud.
3. Increased efficiency: Automation of supply chain processes reduces the time and cost associated with manual processes.
4. Improved traceability: Blockchain allows for the tracking of products from the source to the end-user, improving traceability and accountability.

Conclusion: Blockchain with smart contracts can provide a secure and transparent platform for supply chain management. By implementing this technology, companies can improve the efficiency, transparency, and traceability of their supply chain processes. The use of blockchain with smart contracts is becoming increasingly popular in various industries, and it has the potential to revolutionize the way we manage and track transactions in the supply chain.

Sample Smart contract:

```
pragma solidity ^0.8.0;

contract SupplyChain {
struct Product {
string name;
uint256 quantity;
bool verified;
}
mapping (address => Product) public products;
address payable public logisticsProvider;
uint256 public totalPayment;
constructor(address payable _logisticsProvider, uint256 _totalPayment) {
logisticsProvider = _logisticsProvider;
totalPayment = _totalPayment;
}
function addProduct(string memory _name, uint256 _quantity) public {
products[msg.sender] = Product(_name, _quantity, false);
}
function verifyProduct(address _supplier) public {
```

```
require(products[_supplier].quantity > 0, "No product found for the supplier");
products[_supplier].verified = true;
logisticsProvider.transfer(totalPayment);
}
}
```

Screenshots: