Varendra Maurya
200010055

Important modules and implementations of functionalities of the DApp

- **Payment.sol**

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ≥0.7.0 <0.9.0;

contract payment{


    address payable owner; //owner is going to receive funds
    constructor(){
        owner = payable(msg.sender);
    }

    function buyItem() external payable{
        require(msg.value>0,"Please pay more than 0 ether");
        uint256 balance = address(msg.sender).balance;
        uint256 balanceInEther = balance / 1 ether;
        require(balanceInEther > msg.value,"Please pay more than 0 ether");
        owner.transfer(msg.value);
    }


}
```

This is a simple payment contract that allows users to buy an item by sending ether to the contract. Here's what the code does:

- **address payable owner**: This variable holds the address of the owner who will receive the funds sent to the contract.
- **constructor()**: This is the constructor function that is called when the contract is deployed. It sets the owner variable to the address of the contract deployer.
- **function buyItem() external payable**: This is the main function of the contract that is called when a user wants to buy an item. It is marked as payable, which means it can receive ether. It also has the external modifier, which means it can only be called from outside the contract.
- **require(msg.value > 0, "Please pay more than 0 ether"):** This line checks that the value sent with the transaction is greater than 0. If it is not, the function will revert and the transaction will fail.
- **uint256 balance = address(msg.sender).balance**: This line retrieves the balance of the sender's address and stores it in the balance variable.
- **uint256 balanceInEther = balance / 1 ether**: This line converts the balance to ether and stores it in the balanceInEther variable. Note that 1 ether is equal to 10^18 wei.

- **require(balanceInEther >= msg.value, "Please pay more than the item price")**: This line checks that the balance of the sender's address is greater than or equal to the price of the item. If it is not, the function will revert and the transaction will fail.
- **owner.transfer(msg.value)**: This line transfers the ether sent with the transaction to the owner address. The transfer function is used to send ether from the contract to an external address.

Deploy.js

```
const hre = require('hardhat');

async function main() {
  const Payment = await hre.ethers.getContractFactory('payment'); //fetching bytecode and ABI
  const payment = await Payment.deploy(); //creating an instance of our smart contract

  await payment.deployed(); //deploying your smart contract

  console.log('Deployed contract address:', `${payment.address}`);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

- **const hre = require('hardhat');**: This line imports the Hardhat library to interact with Ethereum.
- **async function main() { ... }**: This is the main function of the script, which is marked as async to allow the use of await with Promises.
- **const Payment = await hre.ethers.getContractFactory('payment');**: This line retrieves the compiled bytecode and Application Binary Interface (ABI) of the payment contract using the Hardhat ethers library.
- **const payment = await Payment.deploy();**: This line creates an instance of the payment contract and deploys it to the local Ethereum network. The deploy function creates a transaction object to deploy the contract and returns a Promise.
- **await payment.deployed();**: This line waits for the contract to be deployed and confirms that the contract has been deployed successfully. It returns a Promise that resolves to the deployed contract instance.
- **console.log('Deployed contract address:', ${payment.address})**;: This line logs the address of the deployed contract instance to the console.
- **main().catch((error) => { ... })**: This line calls the main function and catches any errors that occur during execution. If an error occurs, it logs the error to the console and sets the exit code of the process to 1.

Payment initiation from frontend and purchase of items

```
var state = {
        provider: null,
        signer: null,
        contract: null,
    };
    const template = async (add) => {
        const contractAddres = `${add}`;
        const contractABI = abi.abi;
```

```
        try {
            const { ethereum } = window;
            const account = await ethereum.request({
                method: 'eth_requestAccounts',
            });
            // const accounts = await

            window.ethereum.on('accountsChanged', () => {
                window.location.reload();
            });
            setAccount(account);
            const provider = new ethers.providers.Web3Provider(ethereum); //read the
Blockchain
            const signer = provider.getSigner(); //write the blockchain

            const contract = new ethers.Contract(contractAddres, contractABI, signer);
            console.log(contract);

            state = { provider, signer, contract };
            console.log({ provider, signer, contract });

            window.localStorage.setItem('state', state);
            console.log(localStorage.getItem('state'));
        } catch (error) {
            console.log(error);
        }
    };
    const buy = async (event, add, price) => {
        event.preventDefault();
        await template(add);
        const { contract } = state;
        const amount = { value: ethers.utils.parseEther(`${price}`) };
        const transaction = await contract.buyItem(amount);

        await transaction.wait();
        console.log(transaction);
        // alert('Transaction is successul');
        axios
            .post('http://localhost:4000/api/buy/', {
                itemId: el._id,
                price: price,
                user: localStorage.getItem('email'),
                paymentId: transaction.hash,
            })
            .then((res) => {
                console.log(res);
```

```
                alert(
                    'Transaction is successul! Please find your item in your profile.'
                );
            })
            .catch((err) => {
                console.log(err);
            });
    };
```

Here is the breakdown of the implementation of item purchase and payment.

1. The user clicks the buy button on a product listing in the web application.
2. The buy function in the smart contract is called to initiate the payment process.
3. The template function is called to establish a connection with the user's Metamask account. Metamask is a browser extension that allows users to interact with Ethereum-based applications and manage their accounts.
4. Once the connection is established, the requested amount is paid using the user's Ethereum account, accessed through Metamask.
5. The transaction hash and details of the purchased item are stored in a MongoDB database. The transaction hash is a unique identifier for the transaction on the Ethereum blockchain, and the details of the purchased item might include information such as the name, price, and quantity of the item.
6. The user can later view their transaction history by retrieving the transaction details from the MongoDB database.

Overall, this process allows users to make payments for items using their Ethereum accounts and ensures that the transaction details are stored securely in a database for future reference.

- Fetching the block details from ganache server

```
// Connect to local Ganache instance
        const web3 = new Web3('http://localhost:7545');

        // Get transaction details
        const tx = await web3.eth.getTransaction(txHash);
        console.log(tx);
        // Get block details
        const block_ = await web3.eth.getBlock(tx.blockNumber);
        setBlock(block_);
        console.log(block_);
```

This is the implementation of fetching the block and transaction details from the ganache server of particular purchase. txHash is the hash (transactionId) generated during the payment/purchase transaction, we use this hash to fetch the transaction, and using its block number, we fetch all the block details using web3 functions