StyleElite

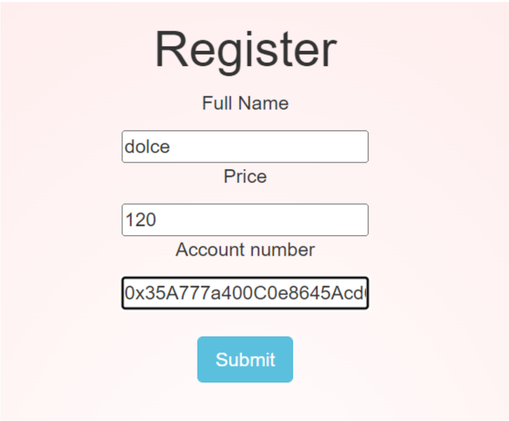
Prepared by:

Akanksha Reddy Yedla | ayedla@buffalo.edu | 50360242
Arshabh Semwal | arshabhs@buffalo.edu | 50419031

The idea is to give a chance to our users and help them in leading the fashion style around them. This project allows a user to become the first few persons to adopt the latest fashion trends straight from the world's best designer's latest and upcoming lines of clothing and accessories. Users can browse through the website and can select one or multiple designer lines. Anybody can register themselves as designers and can market and sell their products on StyleElite. The buyers can select their favorite designer and order their mystery box. Next… you wait for the **stylebox** to arrive which will contain the latest designer clothing and accessories based on your budget and choice.

Major Functionalities

Register: Anybody can register themselves as designers by providing their details like name, price of their mystery box and account number. All the details entered are stored in the blockchain. Once the registration is done the designer's products are displayed on the website.



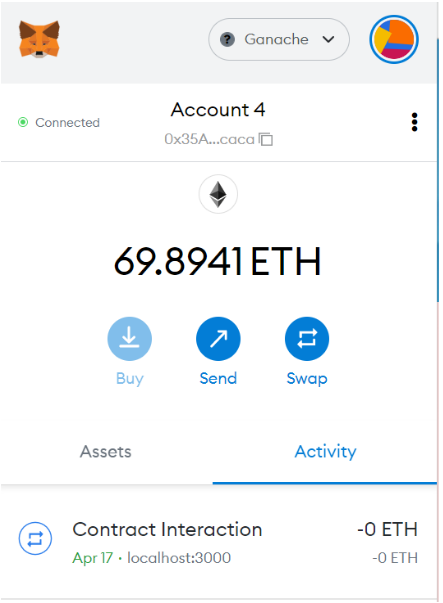
Register

Full Name: dolce

Price: 120

Account number: 0x35A777a400C0e8645Acd

Submit



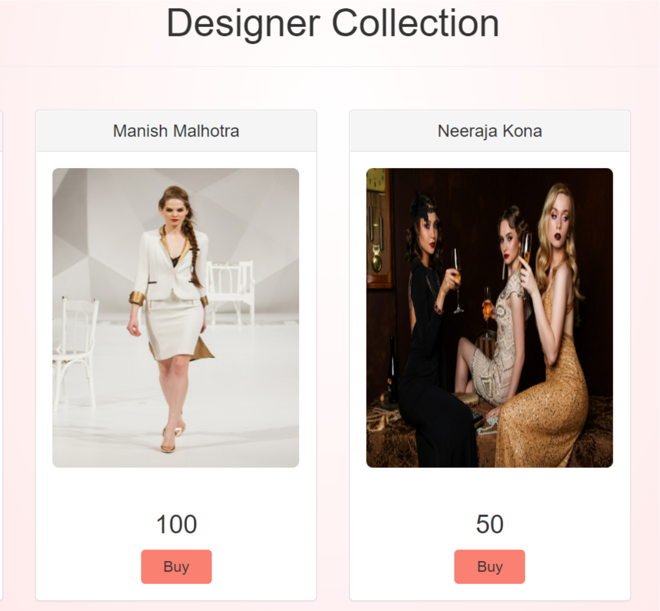
Connected

Account 4
0x35A...caca

69.8941 ETH

Contract Interaction -0 ETH
Apr 17 · localhost:3000 -0 ETH

Buy: A buyer can directly buy the mystery box from their favorite designer. The transaction takes place from buyer's account to seller's account and it is recorded in the blockchain.



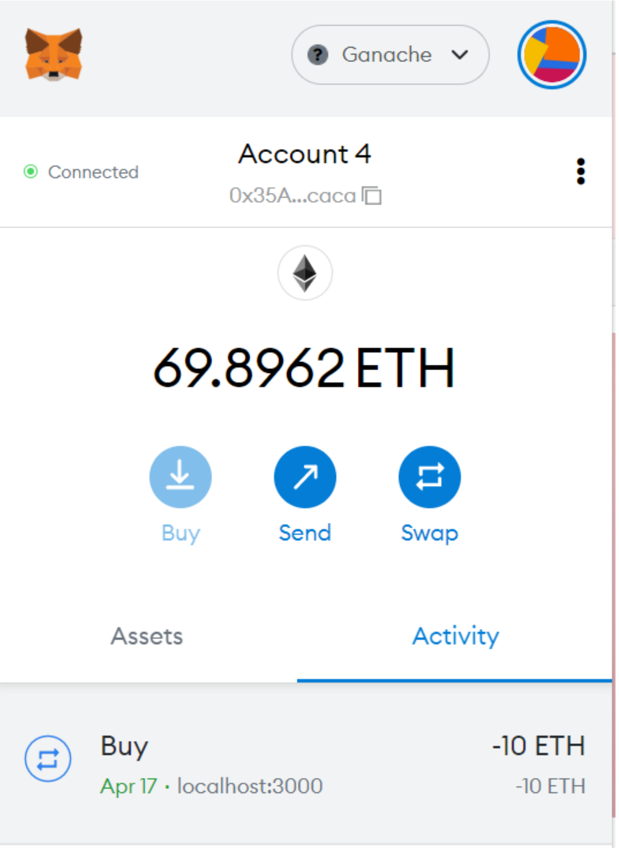
Designer Collection

Manish Malhotra

100

Neeraja Kona

50



Ganache

Connected

Account 4

0x35A...caca

69.8962 ETH

Buy
Apr 17 · localhost:3000
-10 ETH
-10 ETH

Remove Account: The admin can remove any account from the website.

Block Account: The admin has the right to block any account temporarily.

Unblock Account: The admin has the right to unblock any account.

Instructions to test, deploy and interact

First open Ganache and add the ganache network to metamask. Once ganache and metamask setup is complete follow the instructions.

To compile the contract first open the folder se-contract. Once you are in the folder open powershell or terminal and write the command as follows:

- truffle compile
- truffle migrate

Once the code is successfully migrated to the Ganache local blockchain network, the go to the folder se-app. Once inside se-app folder open terminal/powershell and write the following command:

- npm start

The server will start on localhost:3000. Open this link in the browser an you can see the StyleElite dapp running here.

In the dapp you can see register form and 5 designer item already present. These 5 items are for show only. To perform buy operation first you need to register one or more designers. Once you register one or more designer you can either buy, block, unblock or add admin based on the account you use. Only admins can remove items, add new admin, block an address or remove an item but anyone can register a designer mystery box for sale. The account that migrate the contract to blockchain is automatically made admin.

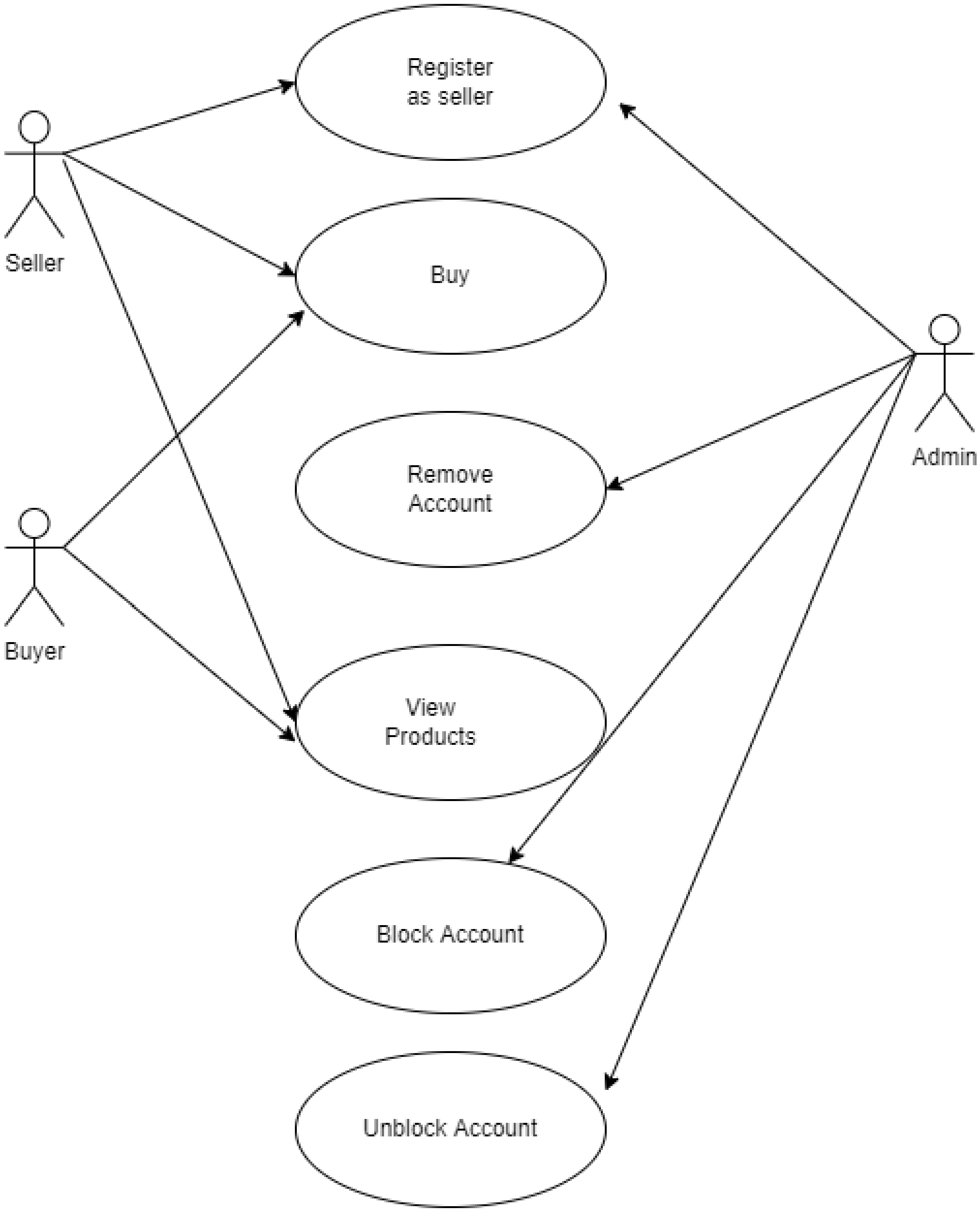
For buy operation you can observe that your account is decrease by a sum of price of item and gas fees and the account you added as seller has its balance increased by the price amount.

Design Diagrams

Quad Chart Diagram

Use case: Problem statement: a decentralized blockchain-based marketplace of designer mystery boxes.	**Issues with existing centralized model** 1. There is no single platform where anybody can register themselves as designers. 2. The customers have to select their outfits without any stylist's assistance. 3. Costs of the products are higher due to the transaction costs of the intermediaries. 4. There is a dependency on third parties for transaction verification which might cause some delay. 5. The transactions are sometimes not safe.
Proposed blockchain based solution 1. A decentralized marketplace is established where designers can sell their curated mystery box. 2. Buyers can directly buy the mystery box from their favorite designer. 3.Seamless payment settlement is enabled where the tokens are transferred from buyer's account to the seller's account. 4. All transactions are recorded on blockchain.	**Benefits** 1. Provides the designers with a platform to market and sell their products. 2. Payment settlement solution that is safe and easy to use. 3. Products are cheaper as there will be no transaction costs of the intermediaries. 4. Potential for a token-based business model. 5. An innovative model for the fashion industry as a whole. 6. User friendly interface.

Use case Diagram



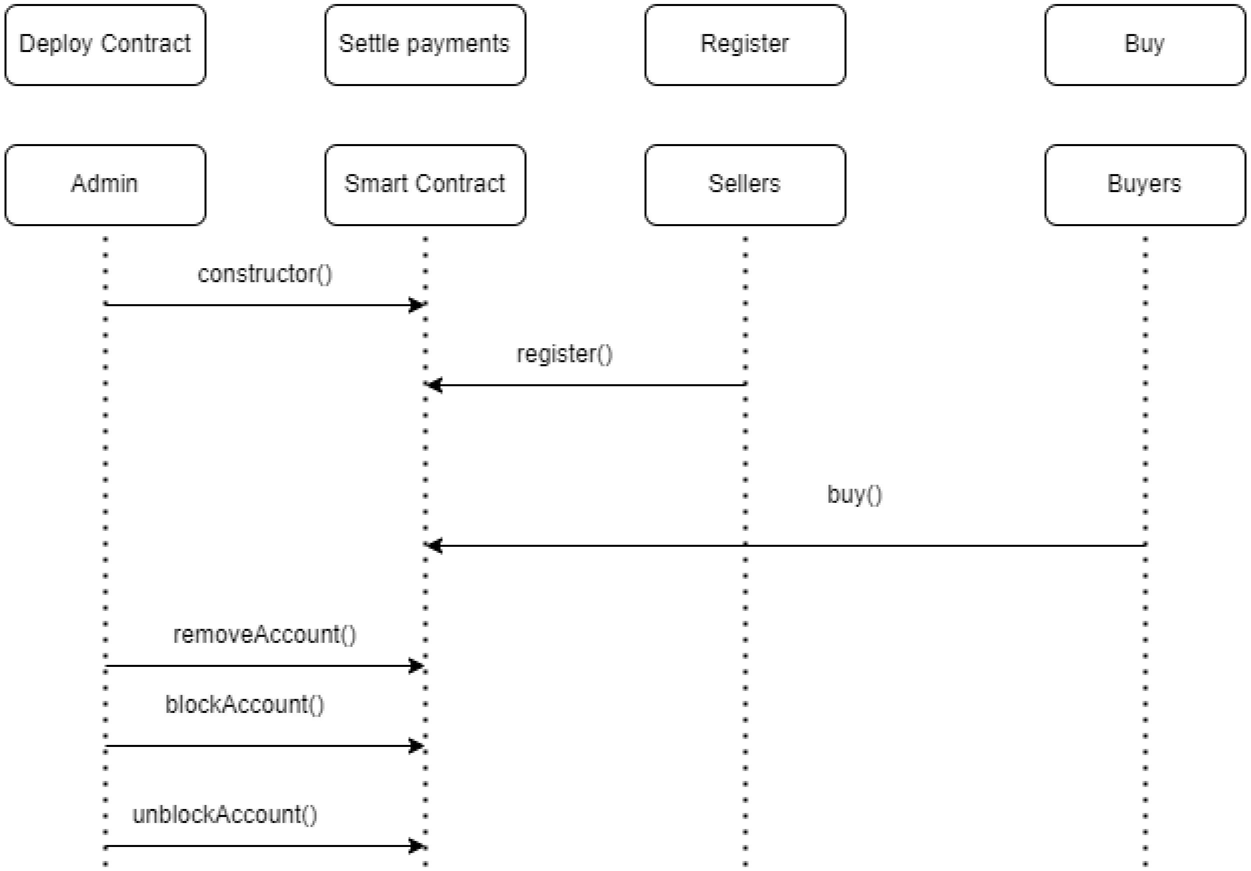
Contract Diagram

StyleElite
struct SellerItem{ } SellerItem[] sellerItem mapping(address => bool) public seller mapping(address => bool) private admin

modifier OnlyAdmin(){   }
modifier sufficientMoneySent(uint itemValue){ }
modifier SellerActive(address sellerAdd){   }

constructor() public payable{ }
function addAdmin(address adminAddr) public OnlyAdmin{ }
function buy(address payable itemSeller, uint itemPrice) public payable sufficientMoneySent(itemPrice) SellerActive(itemSeller){   }
function removeAccount(address sellerAddr) public OnlyAdmin{   }
function blockAccount(address sellerAddr) private OnlyAdmin{   }
function unblockAccount(address sellerAddr) private OnlyAdmin{   }
function getBalanceContract() public view returns(uint){   }
function registerSeller(uint id_, uint price_, address sellerAddr) public returns(uint, uint, address){   }
function getLength() public view returns(uint){   }
function getItemsByIndex(uint index)public view returns(uint, uint, address){   }

Sequence Diagram



Solidity Code

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract StyleElite {

    struct SellerItem{
        uint id;
        uint price;
        address accountNumber;
    }

    SellerItem[] sellerItem;
    mapping(address => bool) public seller;
    mapping(address => bool) private admin;

    constructor() public payable{
        admin[msg.sender] = true;
    }
    modifier OnlyAdmin(){
        require(admin[msg.sender]);
        _;
    }

    modifier sufficientMoneySent(uint itemValue){
        require(msg.value>=itemValue);
        _;
    }

    modifier SellerActive(address sellerAdd){
        require(seller[sellerAdd]);
        _;
    }

    function addAdmin(address adminAddr) public OnlyAdmin{
        admin[adminAddr] = true;
    }
    function buy(address payable itemSeller, uint itemPrice) public payable
sufficientMoneySent(itemPrice) SellerActive(itemSeller){
        itemSeller.transfer(msg.value);
    }
    function removeAccount(address sellerAddr) public OnlyAdmin{
        delete seller[sellerAddr];
    }
    function blockAccount(address sellerAddr) private OnlyAdmin{
        seller[sellerAddr] = false;
    }
    function unblockAccount(address sellerAddr) private OnlyAdmin{
        seller[sellerAddr] = true;
    }
    function getBalanceContract() public view returns(uint){
        return address(this).balance;
    }

    function registerSeller(uint id_, uint price_, address sellerAddr) public returns(uint,
uint, address){//OnlyAdmin
        seller[sellerAddr] = true;
        SellerItem memory item;
        item.id=id_;
        item.price=price_;
        item.accountNumber=sellerAddr;
        sellerItem.push(item);
        uint index = sellerItem.length-1;
        return
(sellerItem[index].id,sellerItem[index].price,sellerItem[index].accountNumber);
    }

    function getLength() public view returns(uint){
        return sellerItem.length;
    }
    function getItemsByIndex(uint index)public view returns(uint, uint, address){
        return
(sellerItem[index].id,sellerItem[index].price,sellerItem[index].accountNumber);
    }

}
```

Data structures

struct SellerItem{ }: The structure SellerItem has details of the designer which includes id,price and account number.

SellerItem[] sellerItem: An array of SellerItems which has the details of all the designers.

mapping(address => bool) public seller: Mapping of seller's address to boolean value which indicates whether the designer is currently a seller or not.

mapping(address => bool) private admin: Mapping of address to boolean value which indicates whether the designer is currently a admin or not.

Modifiers

modifier OnlyAdmin(){ }: A modifier for OnlyAdmin rule.

modifier sufficientMoneySent(uint itemValue){ }: A modifier to check if the buyer has sufficient balance

modifier SellerActive(address sellerAdd){ }: A modifier to check if the seller is active.

Functions

function addAdmin(address adminAddr) public OnlyAdmin{ }: This function is used to add the admin.

function removeAccount(address sellerAddr) public OnlyAdmin{ }:This function is used to remove any account and it can only be invoked by the admin

function blockAccount(address sellerAddr) private OnlyAdmin{ }: This function is used to block any account and it can only be invoked by the admin

function unblockAccount(address sellerAddr) private OnlyAdmin{ }: This function is used to unblock any account and it can only be invoked by the admin

function registerSeller(string memory name_, uint price_, address sellerAddr) public returns(string memory, uint, address){ }: The function registerSeller takes the name, price and account number of the designer as parameters and stores them in sellerItem.

function buy(address payable itemSeller) public payable {   }: The function buy is used to transfer the amount from buyer’s account to seller’s account.