SecureSwap

Presented by Ryan C, Maisha M, and Aparna S

Overview

Many people turn to Facebook Marketplace, Craigslist, or eBay to sell their used goods. However, there is always a fear of being scammed, either as a buyer or a seller. There are many stories of people who did not receive the product they wanted, or did not receive the money they were owed for the product. It is easy to say that the second-hand exchange market is not very reliable.

We propose SecureSwap, a secure way of buying and selling goods. SecureSwap would run on a decentralized platform, backed by the Ethereum blockchain. This is a service for both buyers and sellers, ensuring efficient transactions and market agent privacy. With the Ethereum blockchain, we can facilitate issuing payments to sellers and withdrawing money from a buyer’s account. The platform would allow all steps in the buying and selling process to be executed fairly, ensuring that both parties are satisfied with the transaction.

Background

Blockchain, notably Ethereum, revolutionizes online marketplaces. Smart contracts guarantee sellers get paid only upon buyer confirmation, minimizing scams. This decentralized system empowers users, offering transparency, security, and efficient dispute resolution, setting a new standard in digital commerce.

Projective Objective & Methodology

Project Objective

Our platform aims to mediate the buying and selling process for second-hand goods. Particularly, we want to create a decentralized platform in which: Creating a reliable platform, disputing transactions and scam prevention

Methodology

Our platform securely stores product information on an smart contract for prompt payments and employs object detection AI to verify transactions, preventing scams in a decentralized manner.

Objectives:

In Scope:

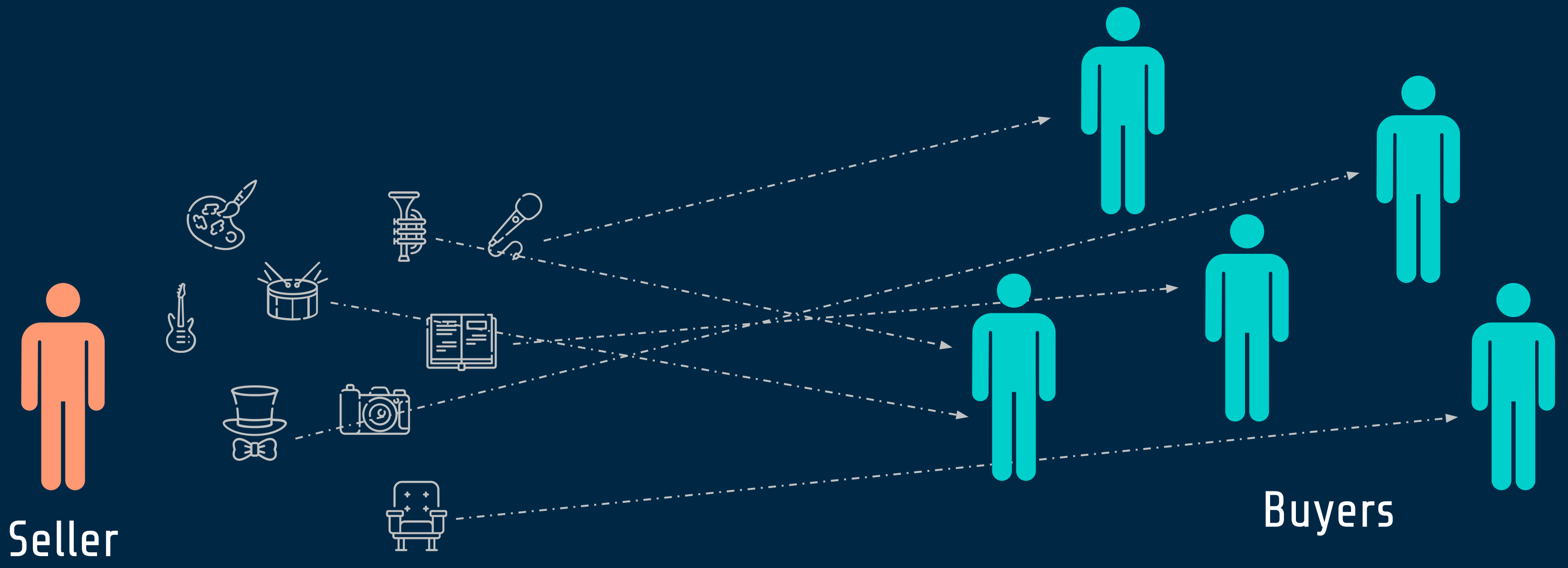
- Create a solidity contract that handles disputes
- We wanted:
  - Have a seller stage a product with images of a product, the price that they wish to sell it for, and stake a fee higher than the value of the product they wish to sell (in case they don’t ship the product, then they will be held accountable). The buyer will stage the appropriate amount of funds into the smart contract that they wish to buy, as well as any personal information.
  - The contract will release the buyer’s address to the seller when the buyer agrees to make a purchase. If all goes well, both parties will approve the transaction, and the contract will send the money to the seller.
- Train an AI Model to do a decentralized “check” of images that users put in

Out of Scope

- The physical swapping of items
- Having the smart contract be able to call the AI on its own (right now users have to manually input pictures)
- Implementing ALL functions from smart contract onto frontend, right now we only have dIsputeTransaction, listProduct, and approveTrasaction on frontend.

Architecture

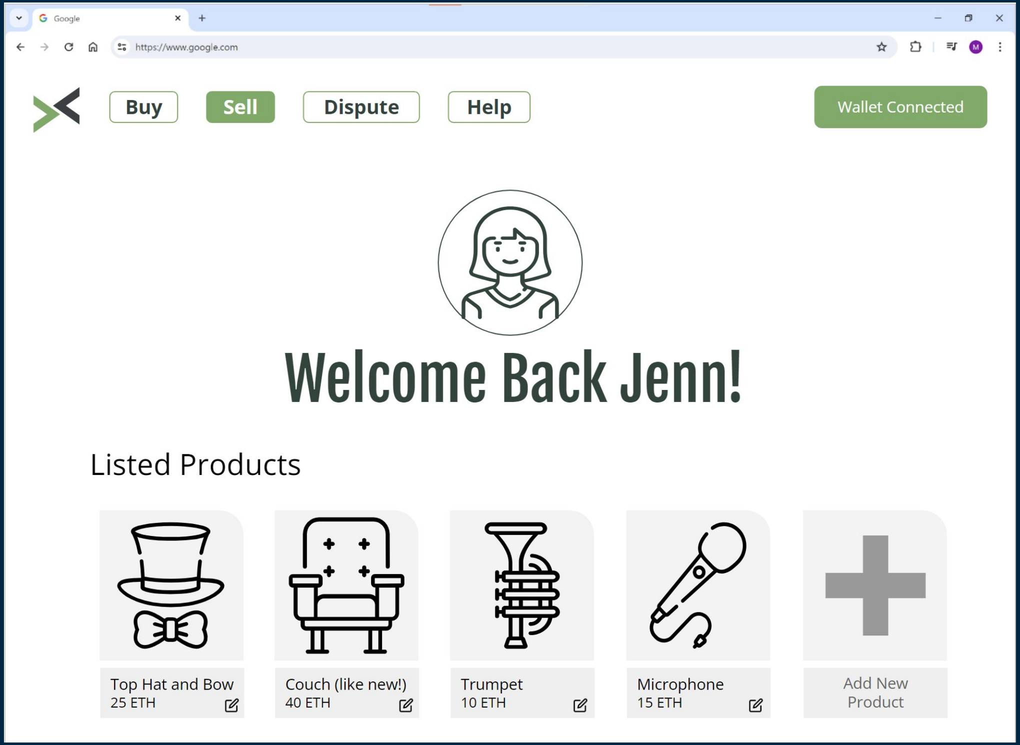
Architecture and Methodology



Seller has items they don’t use anymore and want to sell.

Architecture and Methodology

Seller



Welcome Back Jenn!

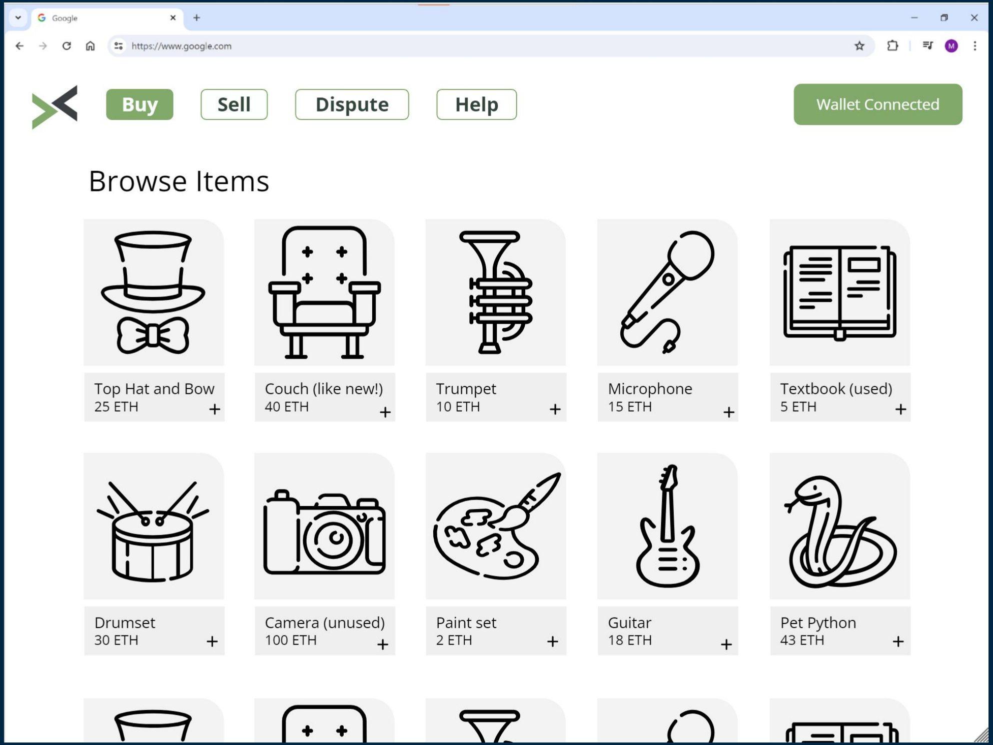
Listed Products

Top Hat and Bow 25 ETH	Couch (like new!) 40 ETH	Trumpet 10 ETH	Microphone 15 ETH	Add New Product

Seller lists items on SecureSwap

Architecture and Methodology

Buyer



Browse Items

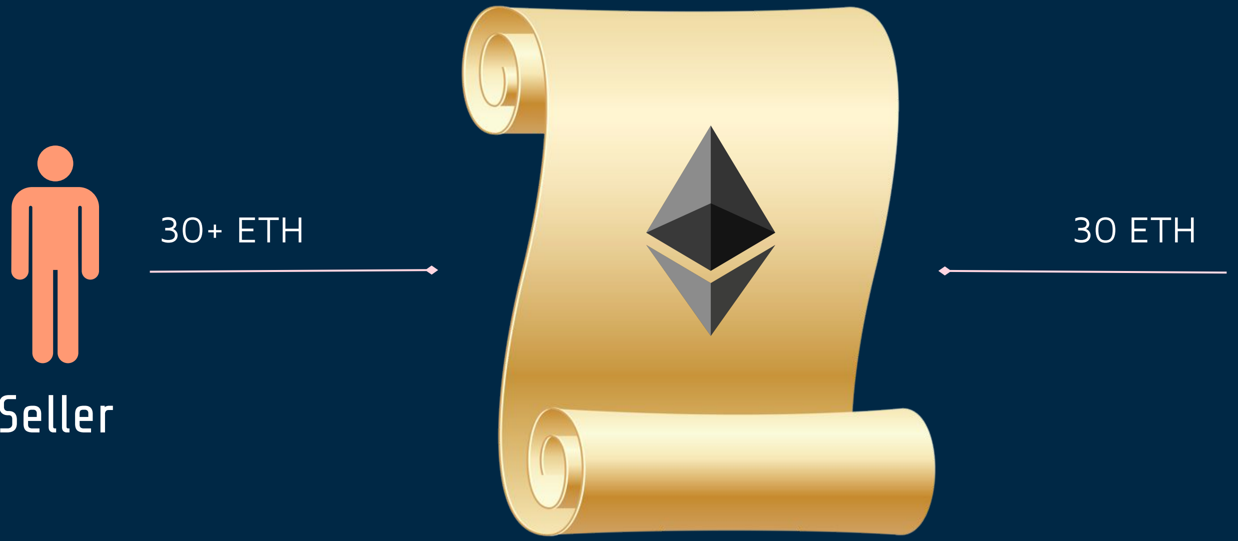
Top Hat and Bow 25 ETH	Couch (like new!) 40 ETH	Trumpet 10 ETH	Microphone 15 ETH	Textbook (used) 5 ETH
Drumset 30 ETH	Camera (unused) 100 ETH	Paint set 2 ETH	Guitar 18 ETH	Pet Python 43 ETH

Buyer looks for products to buy

Architecture and Methodology

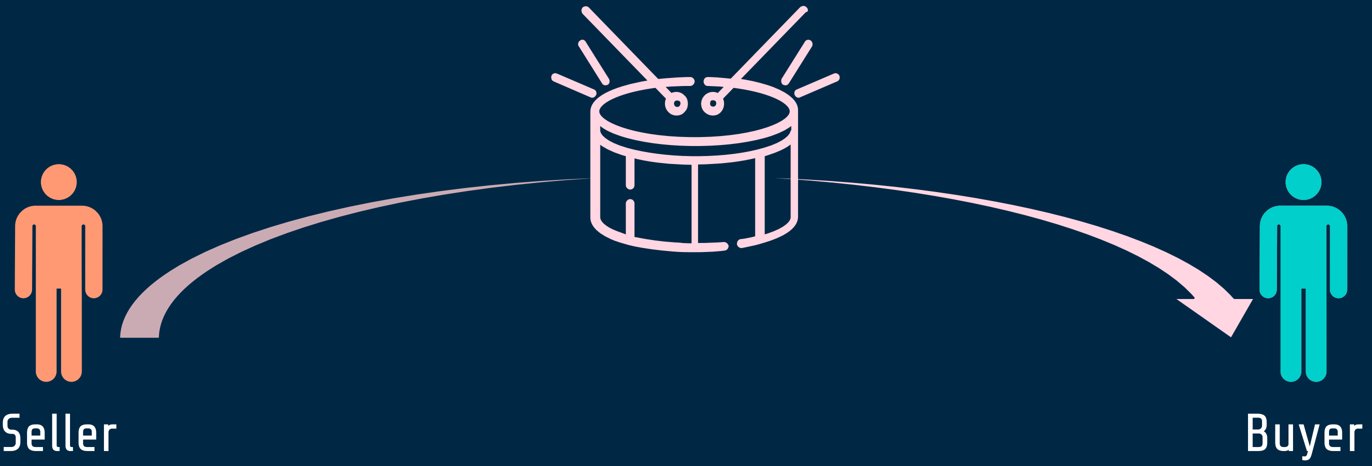
Transaction Begins!

Architecture and Methodology



Money gets staked to the contract

Architecture and Methodology

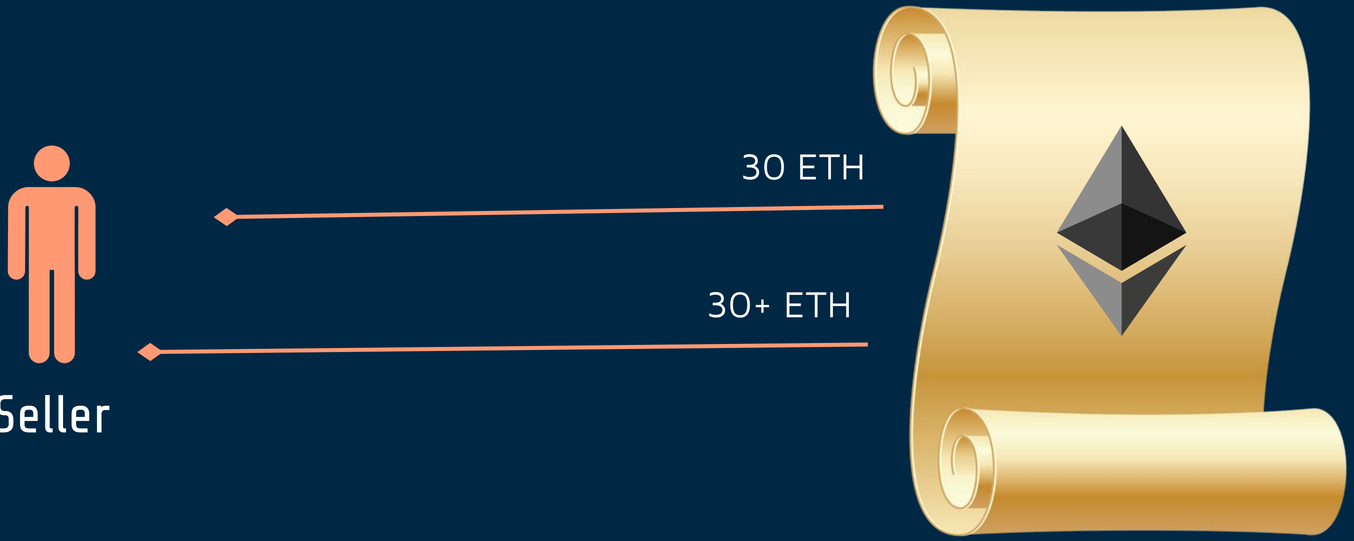


Item gets shipped to buyer

Architecture and Methodology

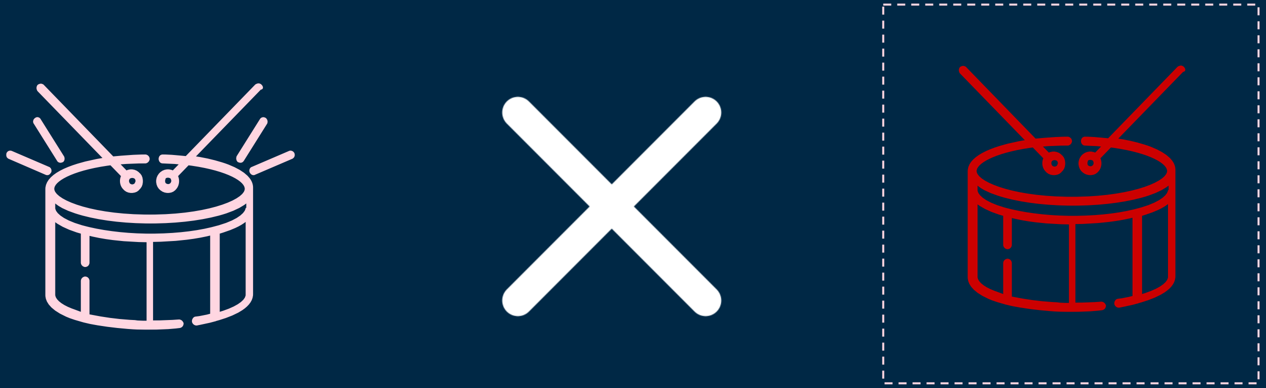
Buyer uploads photo, backend checks if they are the same

Architecture and Methodology



Seller get the buyer’s payment and their deposit back

Architecture and Methodology



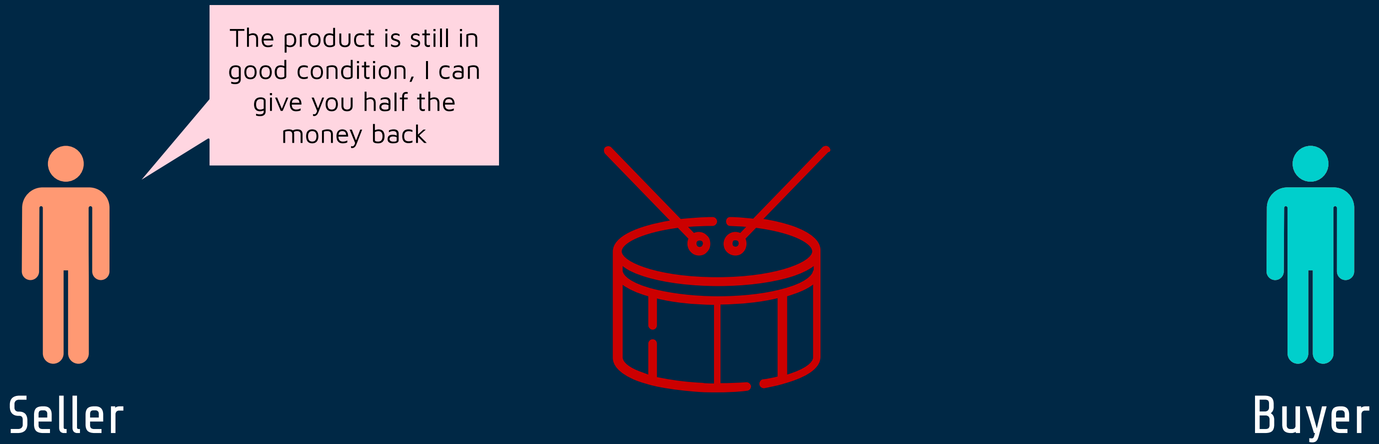
Buyer uploads photo, backend finds they are not the same

Architecture and Methodology

This is not what I wanted. I want my money back!!
Seller
Buyer

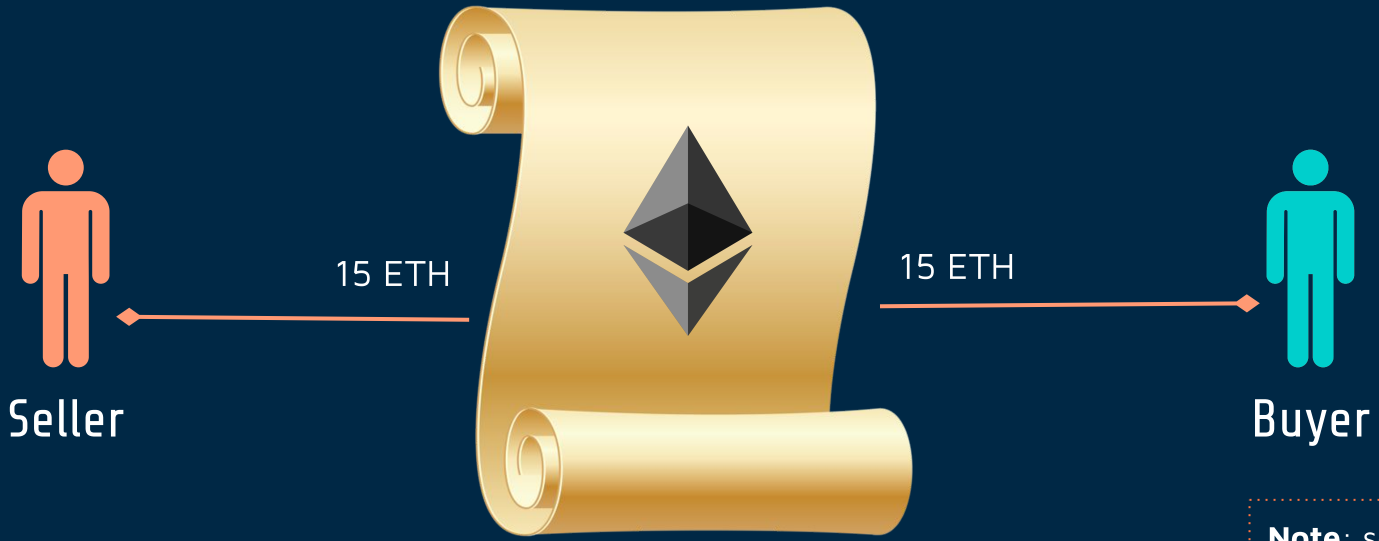
Start a dispute, buyer and seller agree on fair price

Architecture and Methodology


The product is still in good condition, I can give you half the money back
Seller
Buyer

Start a dispute, buyer and seller agree on fair price

Architecture and Methodology

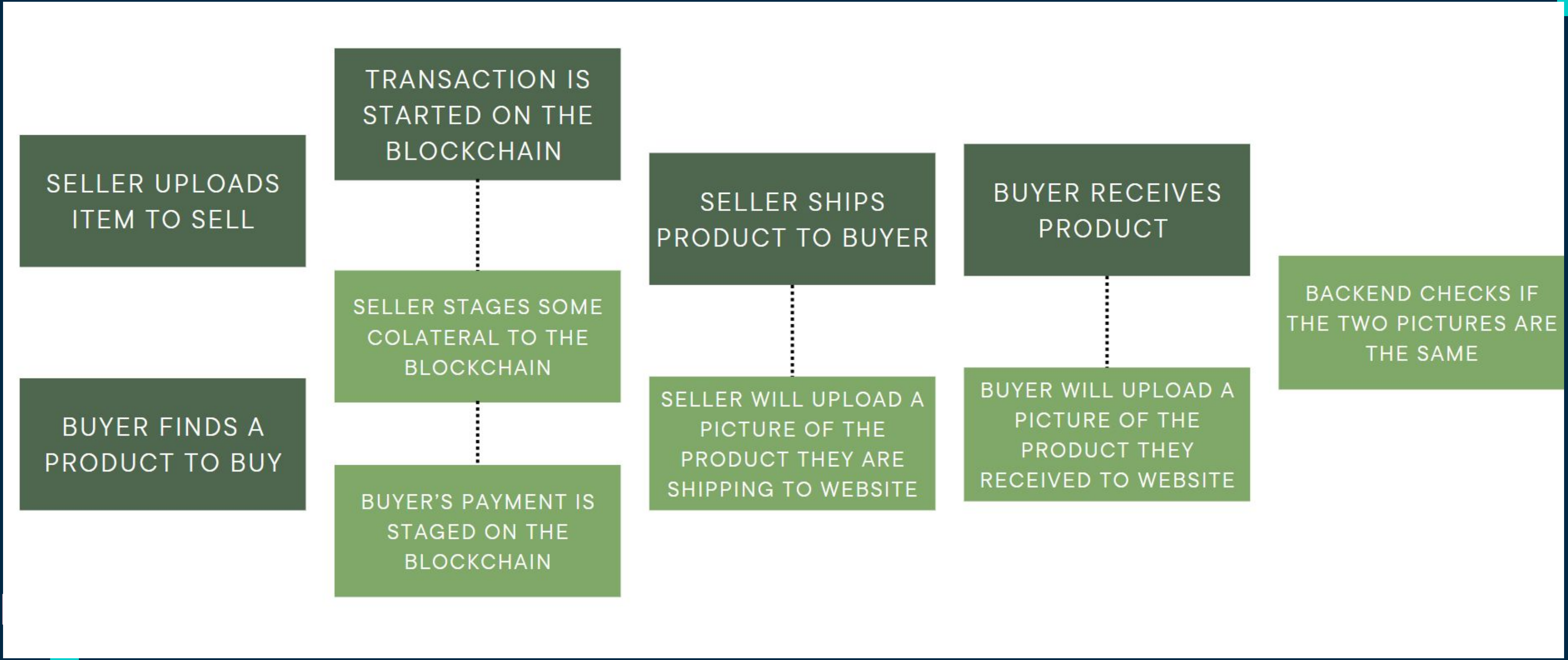


Note: seller does NOT get their deposit back

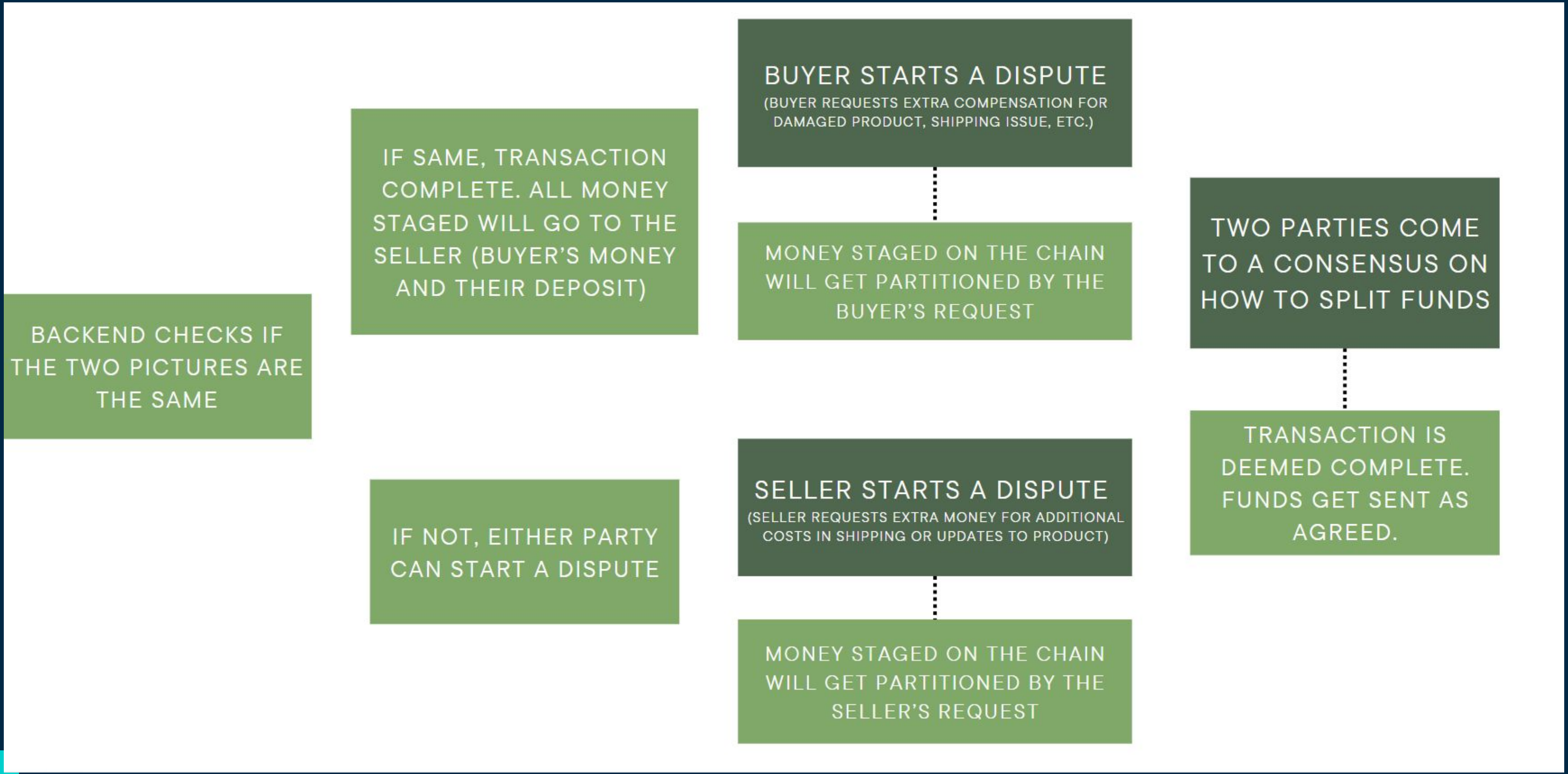
Start a dispute, buyer and seller agree on fair price

Implementation

Implementation


SELLER UPLOADS ITEM TO SELL
BUYER FINDS A PRODUCT TO BUY
TRANSACTION IS STARTED ON THE BLOCKCHAIN
SELLER STAGES SOME COLATERAL TO THE BLOCKCHAIN
BUYER'S PAYMENT IS STAGED ON THE BLOCKCHAIN
SELLER SHIPS PRODUCT TO BUYER
SELLER WILL UPLOAD A PICTURE OF THE PRODUCT THEY ARE SHIPPING TO WEBSITE
BUYER RECEIVES PRODUCT
BUYER WILL UPLOAD A PICTURE OF THE PRODUCT THEY RECEIVED TO WEBSITE
BACKEND CHECKS IF THE TWO PICTURES ARE THE SAME

Implementation

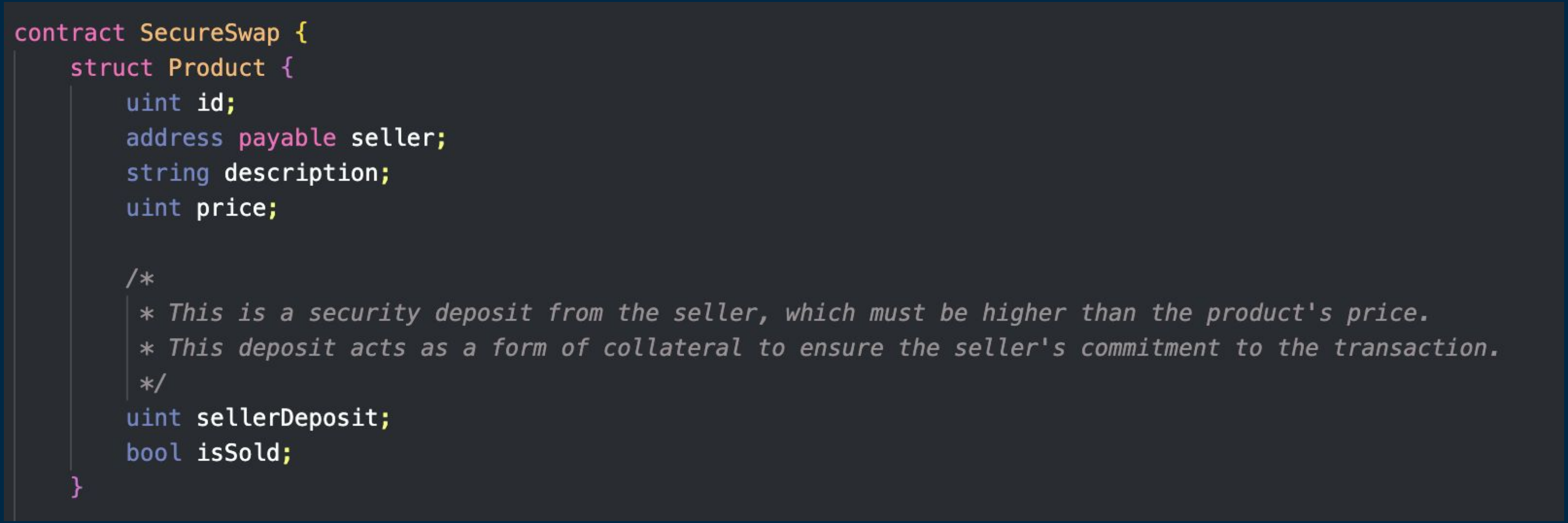

BACKEND CHECKS IF THE TWO PICTURES ARE THE SAME
IF SAME, TRANSACTION COMPLETE. ALL MONEY STAGED WILL GO TO THE SELLER (BUYER'S MONEY AND THEIR DEPOSIT)
IF NOT, EITHER PARTY CAN START A DISPUTE
BUYER STARTS A DISPUTE (BUYER REQUESTS EXTRA COMPENSATION FOR DAMAGED PRODUCT, SHIPPING ISSUE, ETC.)
MONEY STAGED ON THE CHAIN WILL GET PARTITIONED BY THE BUYER'S REQUEST
SELLER STARTS A DISPUTE (SELLER REQUESTS EXTRA MONEY FOR ADDITIONAL COSTS IN SHIPPING OR UPDATES TO PRODUCT)
MONEY STAGED ON THE CHAIN WILL GET PARTITIONED BY THE SELLER'S REQUEST
TWO PARTIES COME TO A CONSENSUS ON HOW TO SPLIT FUNDS
TRANSACTION IS DEEMED COMPLETE. FUNDS GET SENT AS AGREED.

Implementation

SOLIDITY

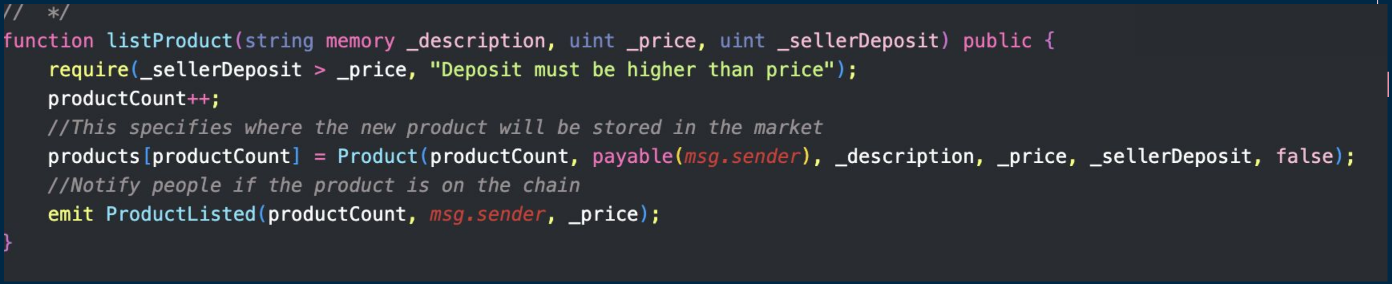
TRUFFLE

useDapp

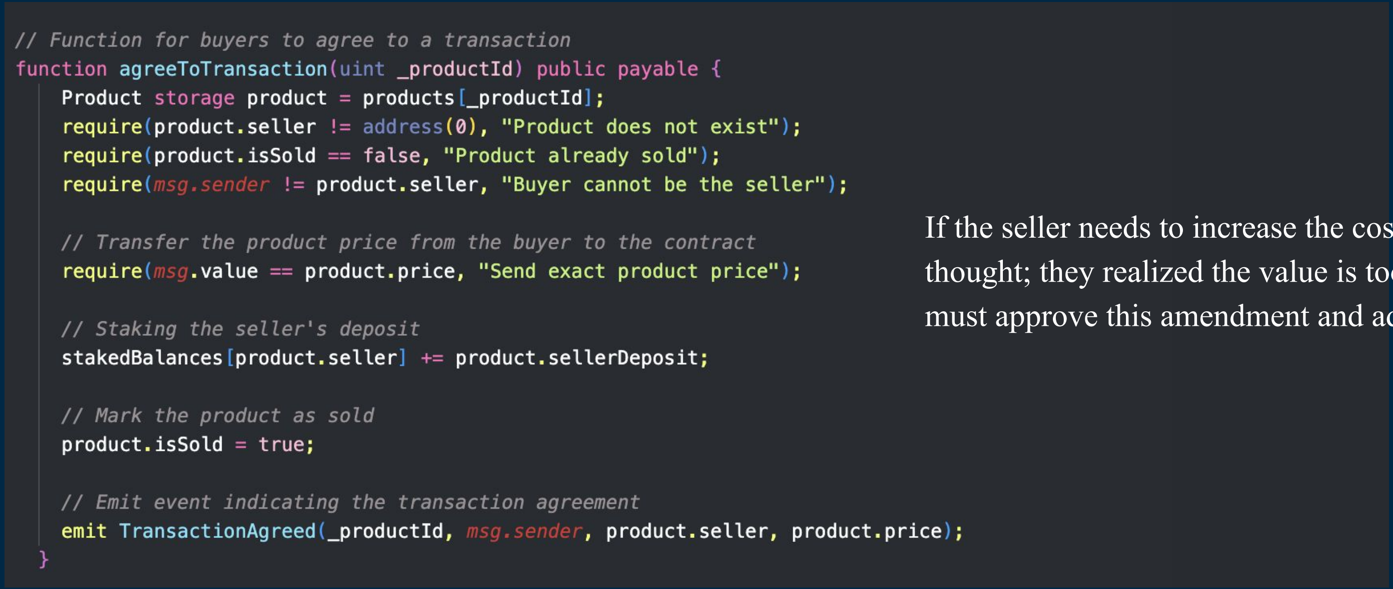


```
contract SecureSwap {
    struct Product {
        uint id;
        address payable seller;
        string description;
        uint price;

        /*
         * This is a security deposit from the seller, which must be higher than the product's price.
         * This deposit acts as a form of collateral to ensure the seller's commitment to the transaction.
         */
        uint sellerDeposit;
        bool isSold;
    }
```

List a product



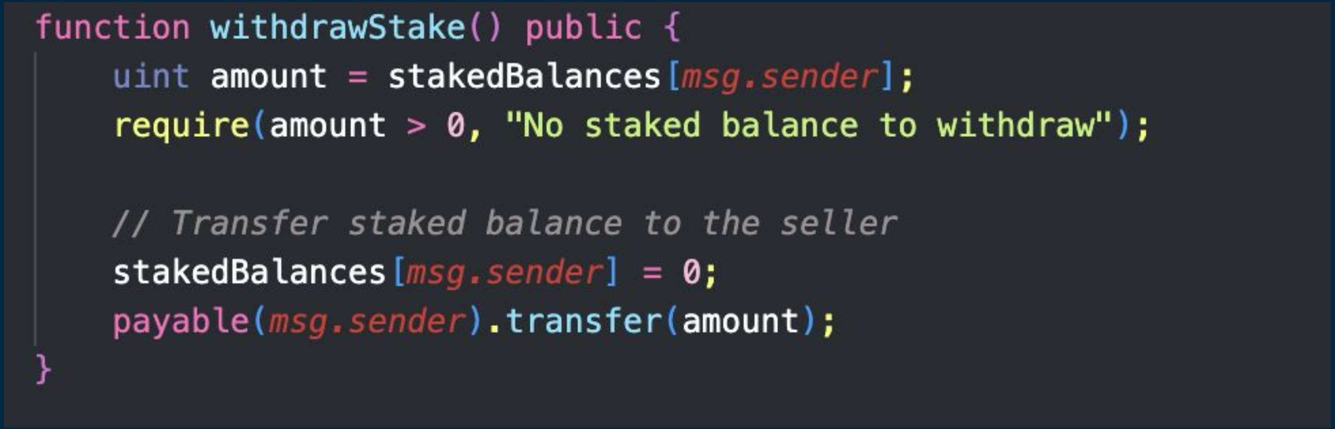
```
//  */
function listProduct(string memory _description, uint _price, uint _sellerDeposit) public {
    require(_sellerDeposit > _price, "Deposit must be higher than price");
    productCount++;
    //This specifies where the new product will be stored in the market
    products[productCount] = Product(productCount, payable(msg.sender), _description, _price, _sellerDeposit, false);
    //Notify people if the product is on the chain
    emit ProductListed(productCount, msg.sender, _price);
}
```



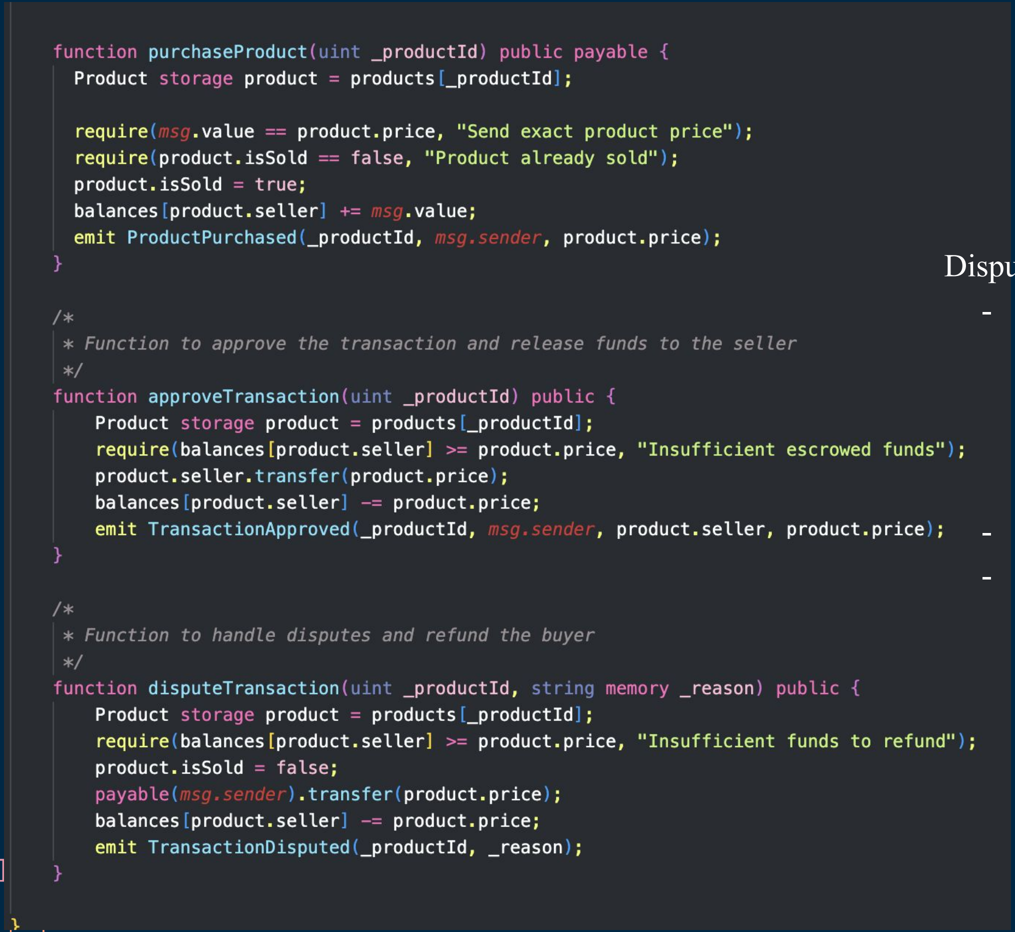
```
// Function for buyers to agree to a transaction
function agreeToTransaction(uint _productId) public payable {
    Product storage product = products[_productId];
    require(product.seller != address(0), "Product does not exist");
    require(product.isSold == false, "Product already sold");
    require(msg.sender != product.seller, "Buyer cannot be the seller");

    // Transfer the product price from the buyer to the contract
    require(msg.value == product.price, "Send exact product price");

    // Staking the seller's deposit
    stakedBalances[product.seller] += product.sellerDeposit;

    // Mark the product as sold
    product.isSold = true;

    // Emit event indicating the transaction agreement
    emit TransactionAgreed(_productId, msg.sender, product.seller, product.price);
  }
```

If the seller needs to increase the cost (shipping is more than they thought; they realized the value is too low, etc.) then the buyer must approve this amendment and add funds to the contract.

If the seller needs to increase the cost and the buyer disagrees with this increase, the seller has the option to end the contract and the contract will send funds back to the buyer automatically.



```
function withdrawStake() public {
    uint amount = stakedBalances[msg.sender];
    require(amount > 0, "No staked balance to withdraw");

    // Transfer staked balance to the seller
    stakedBalances[msg.sender] = 0;
    payable(msg.sender).transfer(amount);
}
```



```
function purchaseProduct(uint _productId) public payable {
  Product storage product = products[_productId];

  require(msg.value == product.price, "Send exact product price");
  require(product.isSold == false, "Product already sold");
  product.isSold = true;
  balances[product.seller] += msg.value;
  emit ProductPurchased(_productId, msg.sender, product.price);
}

/*
 * Function to approve the transaction and release funds to the seller
 */
function approveTransaction(uint _productId) public {
    Product storage product = products[_productId];
    require(balances[product.seller] >= product.price, "Insufficient escrowed funds");
    product.seller.transfer(product.price);
    balances[product.seller] -= product.price;
    emit TransactionApproved(_productId, msg.sender, product.seller, product.price);
}

/*
 * Function to handle disputes and refund the buyer
 */
function disputeTransaction(uint _productId, string memory _reason) public {
    Product storage product = products[_productId];
    require(balances[product.seller] >= product.price, "Insufficient funds to refund");
    product.isSold = false;
    payable(msg.sender).transfer(product.price);
    balances[product.seller] -= product.price;
    emit TransactionDisputed(_productId, _reason);
}

}
```

Dispute Transaction

- If the buyer needs to decrease the sellers cost (product is damaged, the product doesn't work/is not as advertised), the buyer can request a decrease of funds. The seller must approve this, and if the seller agrees then the difference in funds will be sent back from the contract to the buyer.
-
- If the buyer requests to decrease the seller's cost, and the seller doesn't agree with the decrease, the buyer must upload pictures of the malfunctioning product to the AI. The seller has proactively uploaded pictures to the AI which will determine whether the product is, in fact, different from what was advertised. If the AI favors the buyer's story, the contract will refund them, otherwise the seller will get the money in the contract.

Evaluation

Evaluation

Settle Disputes

Evaluate different disputes that occur and resolve them in a decentralized manner

Verify Transaction

AI settles disputes based on picture matching

List and Withdraw Items

Sellers successfully list products to the chain and they are remove when a buyer purchase the item

Demo

Challenges/Successes

Challenges

- Connecting frontend to backend to smart contract
- Having a reliable AI match with tensorflow
- Had to start our website 2x

Successes

- Connecting metamask to the website, being able to have initial functionality
- Our smart contract working for dispute resolution on Remix
- Training tensorflow to recognize different objects

Next Steps

Next Steps

Photo Detection Algorithm

Marketplace UI

Dynamic Deposit Amounts

User accounts

Help & Support Icons

Avatar Icons

Creative Process Icons

Performing Arts Icons

Nature Icons

SEO & Marketing Icons