

CSE 526 Blockchain
Term Project Phase 3
Baby's Essentials

Saurabh Tambolkar – stambolk@buffalo.edu, 50412968
Shantanu Kumar – skumar39@buffalo.edu, 50418500

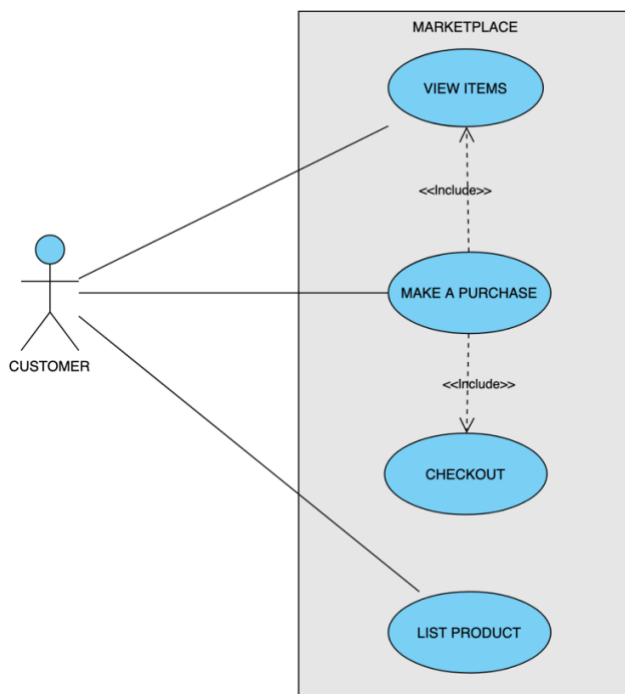
1. Issue Addressed

Through this project, we aim to create a marketplace for products for children like toys, clothing, toiletries etc.

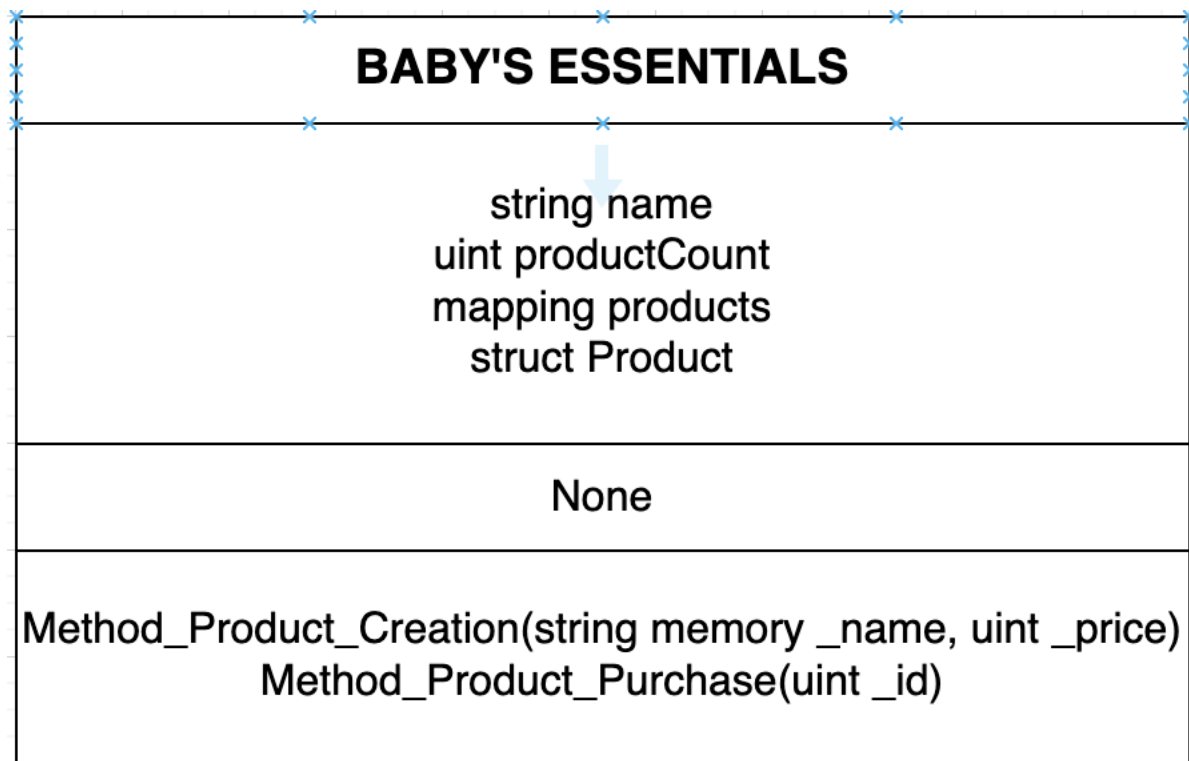
2. Abstract

We plan to create a marketplace that will facilitate its users in buying and selling products related to young children like toys, clothing, toiletries etc. Each user will have his/her separate account. Users will have the facility of buying products listed by other users and the ability to list their own products for sale. While listing a product for sale, each user will list a price for a product along with a description of the product.

3. Design



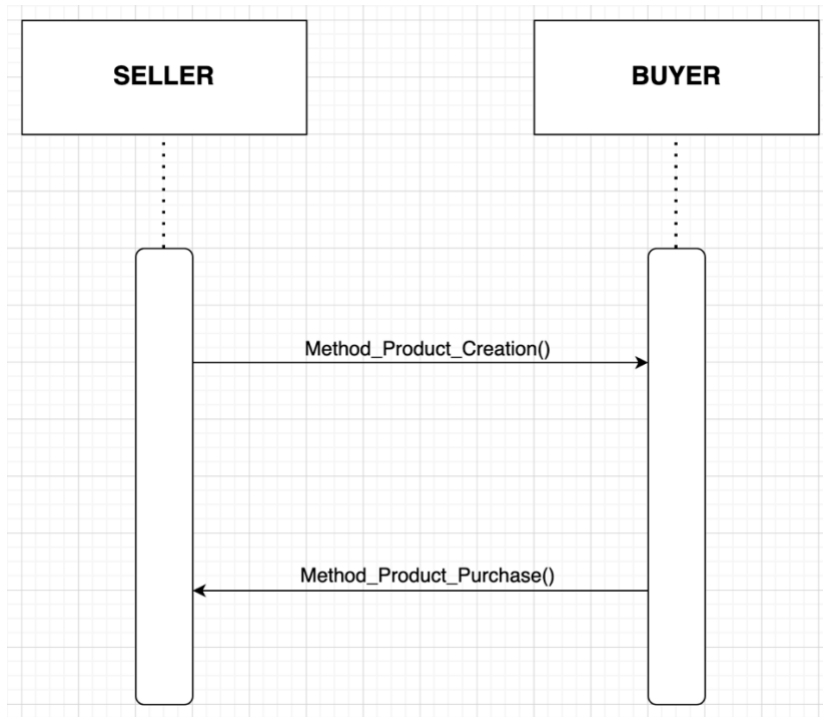
Use case diagram for the marketplace.



Contract diagram.

Use case - Online Marketplace Problem Statement – A decentralized blockchain based marketplace.	Issues with the existing centralized model – 1. Low level of reliability since the marketplace works on a centralized model.
Proposed Blockchain-based solution – 1. Seamless payment method. 2. All transactions recorded on the blockchain for added security. 3. Users can list their own products for sale.	Benefits 1. Saves time, effort, and cost for the customers. 2. Better customer experience. 3. Seamless payment system. 4. Security 5. Privacy

Quad Chart.



Sequence diagram.

4. Smart Contract Functionality

```
struct Product {
    uint id;
    string name;
    uint price;
    address owner;
    bool purchased;
}
```

The definition of a Product. Here, **id** an unsigned integer that defines the identification number of a product, string **name** stores the name of the product, **price** is again an unsigned integer that stores the price of the product, variable **owner** stores the address of the account that currently owns a product, **purchased** is a Boolean variable that stores true if the product has been purchased, false otherwise. It is false by default.

```
mapping(uint => Product) public products;
```

products is a mapping that maps an unsigned integer to a Product struct.

- The unsigned integer signifies the identification number of the product. All products have a unique identification number.
- The purpose of the mapping is that it will allow us to look up a Product by its identification number.

```
uint public productCount = 0;
```

productCount is unsigned integer that stores the total number of products that exist in the smart contract. We use this since it is not possible to check the size of a mapping in Solidity.

```

constructor() public{
    name = "Dapp University Marketplace";
}

```

Our constructor just defines the name of our application.

```

function Method_Product_Creation (string memory prod_name, uint prod_price)
public {
    // _name must be valid.
    require(bytes(prod_name).length > 0);
    // _price must be valid.
    require(prod_price > 0);
    // Increment product count.
    productCount++;
    // Create the product in the mapping.
    products[productCount] = Product(productCount, prod_name, prod_price,
msg.sender, false);
    // Trigger an event
    emit ProductCreated(productCount, prod_name, prod_price, msg.sender,
false);
}

```

Function **Method_Product_Creation** serves the purpose of creating a product. The arguments passed to the function are a string `prod_name` and an unsigned integer `prod_price`. `prod_name` defines the name of the new product and `prod_price` the price of the product.

- In the function, we first check if the name of the product is valid or not. A valid product name in our eyes is any string that has a length > 0.
- We then check if the price of the product is valid. Any positive price is acceptable here.
- Here, we also increment the value of our global variable **productCount** since a new product is being added to the blockchain.
- We then proceed to add the product to our **mapping products**. The owner of the product is the sender of the message which we can retrieve using **msg.sender**. Since the product is newly listed and has not been purchased yet, we set the value of the **purchased** to false.
- In the end, we trigger an event **CreatedProduct**.

```

event CreatedProduct(
    uint id,
    string name,
    uint price,
    address owner,
    bool purchased
);

```

External subscribers can listen for this event to verify that a product was created on the blockchain.

```

function Method_Product_Purchase(uint p_id) public payable {
    // Get the product
    Product memory _product = products[p_id];
    // Get the owner
    address payable _seller = _product.owner;
    // Product should have a valid ID.
    require(_product.id > 0 && _product.id <= productCount);
}

```

```

    // Enough ether must be present.
    require(msg.value >= _product.price);
    // Require that the product has not been purchased already.
    require(!_product.purchased);
    // The buyer and the seller should not be the same.
    require(seller != msg.sender);
    // The ownership of the product should transfer over to the buyer.
    _product.owner = msg.sender;
    // Mark the product as purchased
    _product.purchased = true;
    // Update the state of the product in the mapping products.
    products[p_id] = _product;
    // Pay the seller.
    address(_seller).transfer(msg.value);
    // Trigger an event
    emit ProductPurchased(productCount, _product.name, _product.price,
msg.sender, true);
}

```

Function **Method_Product_Purchase**, as the name suggests, allows the purchase of a product. The only argument passed to the function is **p_id**, which defines the identification number of the purchased product.

- The function is defined as **payable** since involves the transfer of ethers from one account to the other in case of a successful purchase.
- We fetch the product from the mapping **products** and create a copy of it in memory.
- We store the address of the seller in the variable **_seller** since we will require his address to be able to send him the fee for the product.
- Then, we check if the product identification number is valid or not. Also, the identification number must be less than or equal to total product count stored in the variable **productCount**.
- We then check to see if the buyer currency equal to or greater than the price of the product he is trying to purchase and if the buyer is not the seller.
- The ownership of the product is then transferred from the seller to the buyer.
- Variable **purchased** that signifies if the product has been previously purchased or not is then set to true.
- The product is then updated in the mapping **products**.
- A transfer of ethers happens from the buyer to the seller.
- Event **PurchasedProduct** is then triggered.

```

event PurchasedProduct(
    uint id,
    string name,
    uint price,
    address payable owner,
    bool purchased
);

```

PurchasedProduct is like the event **CreatedProduct**.

5. Deployment

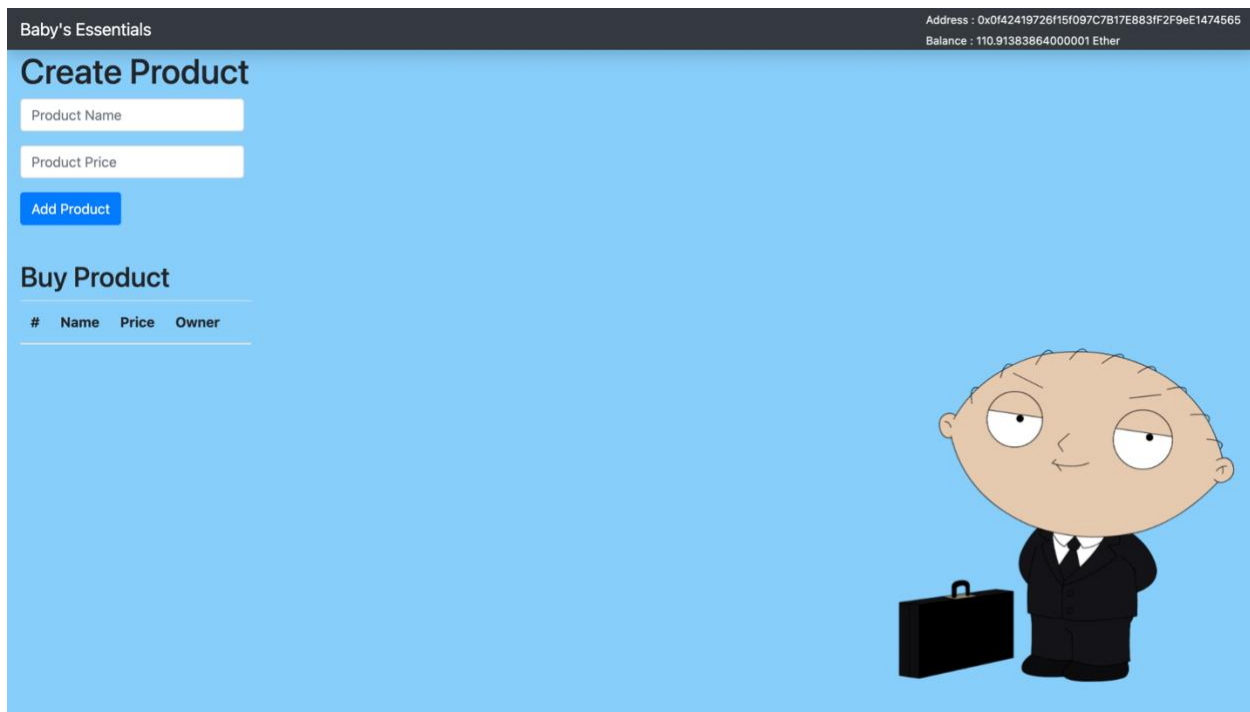
- Extract the contents of the zip file.

- Run a terminal in the same directory and run the below commands in sequence.

```
npm install
truffle compile
truffle test
truffle migrate
truffle run start
```

Note – Our marketplace was tested locally using Ganache.

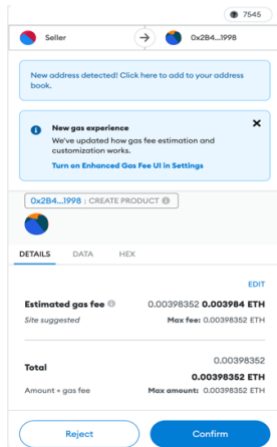
6. Smart Contract Functionality



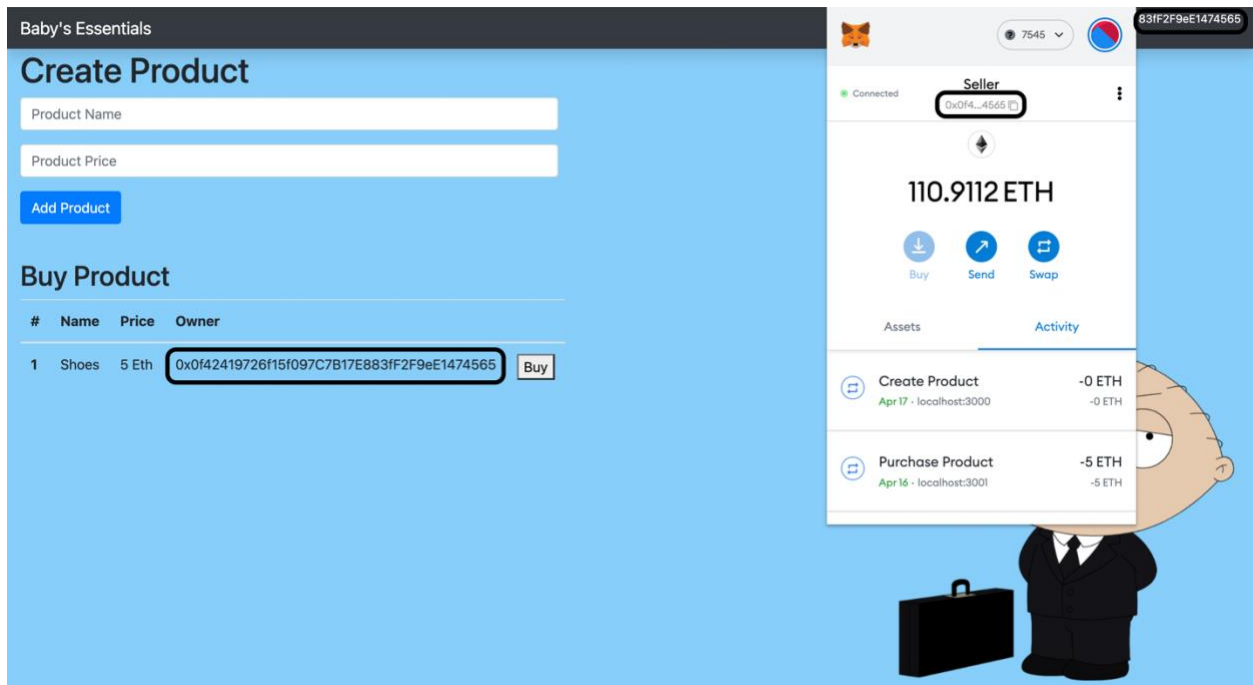
The marketplace when running for the first time.

- On the top left, we have the functionality to add a product. A product will require a name and a price for which it be sold.
- Below the Create Product tab is the list of products that are currently listed. This list will contain the product number, its name, price, and the address of the owner.
- On the top right we have the address and current available balance of the user whose account is currently connected to in Metamask.

6.1 Adding a Product



We get a prompt when we try to add a product.



The marketplace after listing a product for sale.

- We have set up a Metamask account and named it Seller. All the products for sale will be listed using this account for explanatory purposes.
- We have listed an item named "Shoes" for sale having a price of 5 Eth. We see that address listed at the owner column matches that of the address of Seller as displayed in the Metamask page and on the top right of our page.
- Seller currently has a balance of 110.9112 Eth.


Baby's Essentials
Address : 0x0f42419726f15f097C7B17E883f2F9eE1474565
Balance : 110.90647207999999 Ether

Create Product

Add Product

Buy Product

#	Name	Price	Owner	
1	Shoes	5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>
2	Socks	0.5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>
3	Cap	0.5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>



We added a bunch of products.

6.2 Buying a Product

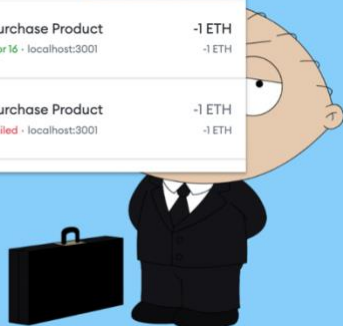
Baby's Essentials
7545
201Ae1cE9120981b

Create Product

Add Product

Buy Product

#	Name	Price	Owner	
1	Shoes	5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>
2	Socks	0.5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>
3	Cap	0.5 Eth	0x0f42419726f15f097C7B17E883f2F9eE1474565	<div style="background-color: #ffc107; padding: 2px 5px; border: 1px solid black;">Buy</div>



7545

Connected Buyer
0xB4C...981b

Buyer

88.9631 ETH

Buy

Send

Swap

Assets
Activity

Purchase Product
Apr 16 · localhost:3001

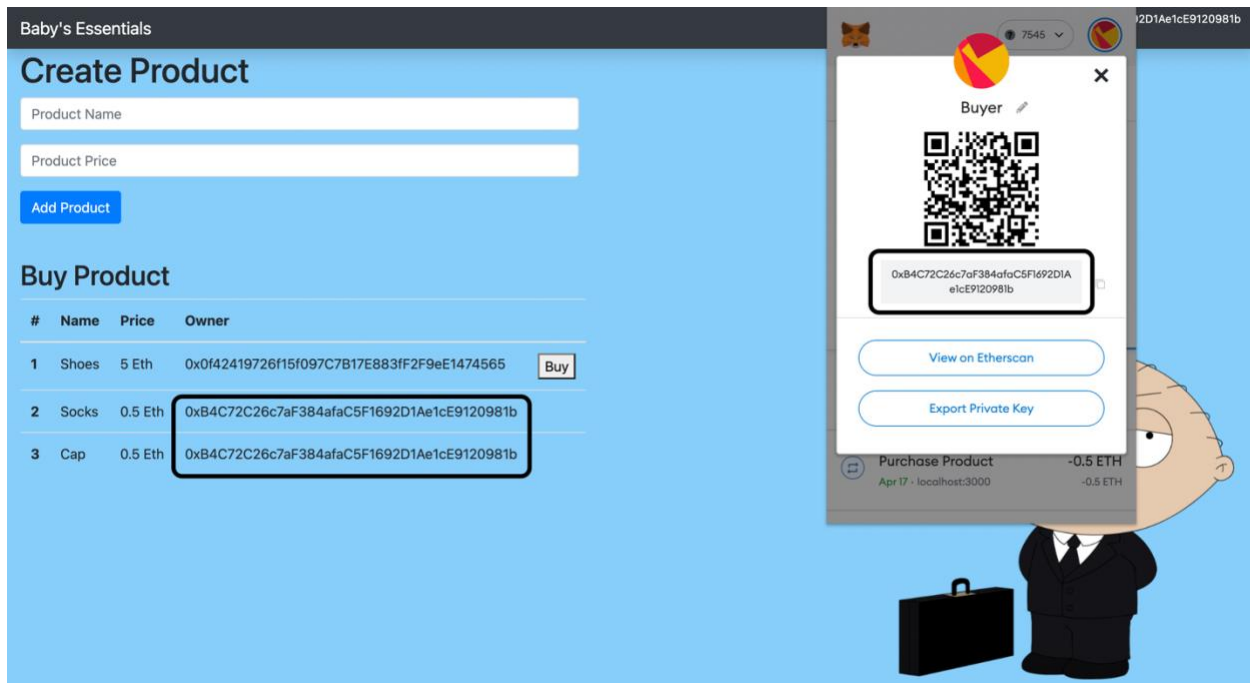
-1 ETH
-1 ETH

Purchase Product
Failed · localhost:3001

-1 ETH
-1 ETH

We now switch over to the Buyer account.

- Buyer currently has a balance of 88.9631 Eth.
- The address displayed in the top right corner matches the address displayed in Metamask.
- We will now buy one or two of the listed products.



We bought two products.

- Buyer bought “Socks” and “Cap”. The balance of Buyer decreased by 1 Eth.
- Buyer now becomes the owner of “Socks” and “Cap”. Also, the Buy button disappears.

Phase 3

7. ERC20 Token

We have utilized the OpenZeppelin Library to implement our custom ERC20 token. Our token has the name **Rinnegan**. We have initialized it using the constructor of the contract ERC20 and assigned 100 Rin to our Primary account.

The various functions of the library are explained below.

`function name() public view virtual override returns (string memory)`

Returns the name of the token.

`function symbol() public view virtual override returns (string memory)`

- Returns the symbol of the token which is a shorter version of the name.

`function totalSupply() public view virtual override returns (uint256)`

- Returns the total supply of the token.

`function balanceOf(address account) public view virtual override returns (uint256)`

- Returns the balance of a specific account.

`function transfer(address to, uint256 amount) public virtual override returns (bool)`

- Transfers the specified number of tokens from one account to the other.

```
function allowance(address owner, address spender) public view virtual override returns (uint256)
```

- Allows account specified by “spender” to spend tokens on behalf of the account specified by the address owner.

```
function transferFrom(
    address from,
    address to,
    uint256 amount
) public virtual override returns (bool)
```

- Transfers tokens equal to “amount” from one account to the other.

```
function _mint(address account, uint256 amount) internal virtual
```

- Assigns “amount” number of tokens to the account having address “account”.

```
function _approve(
    address owner,
    address spender,
    uint256 amount
)
```

- Sets “amount” as the allowance of “spender” over the “owner” s tokens.

7.1 Our Token

Baby's Essentials

Address : 0x440C945e48b703f2E9F1Ece23B3Aa4b86fE410cD
Balance : 200 Rin

Airdrop

Address

Rin

Create Product

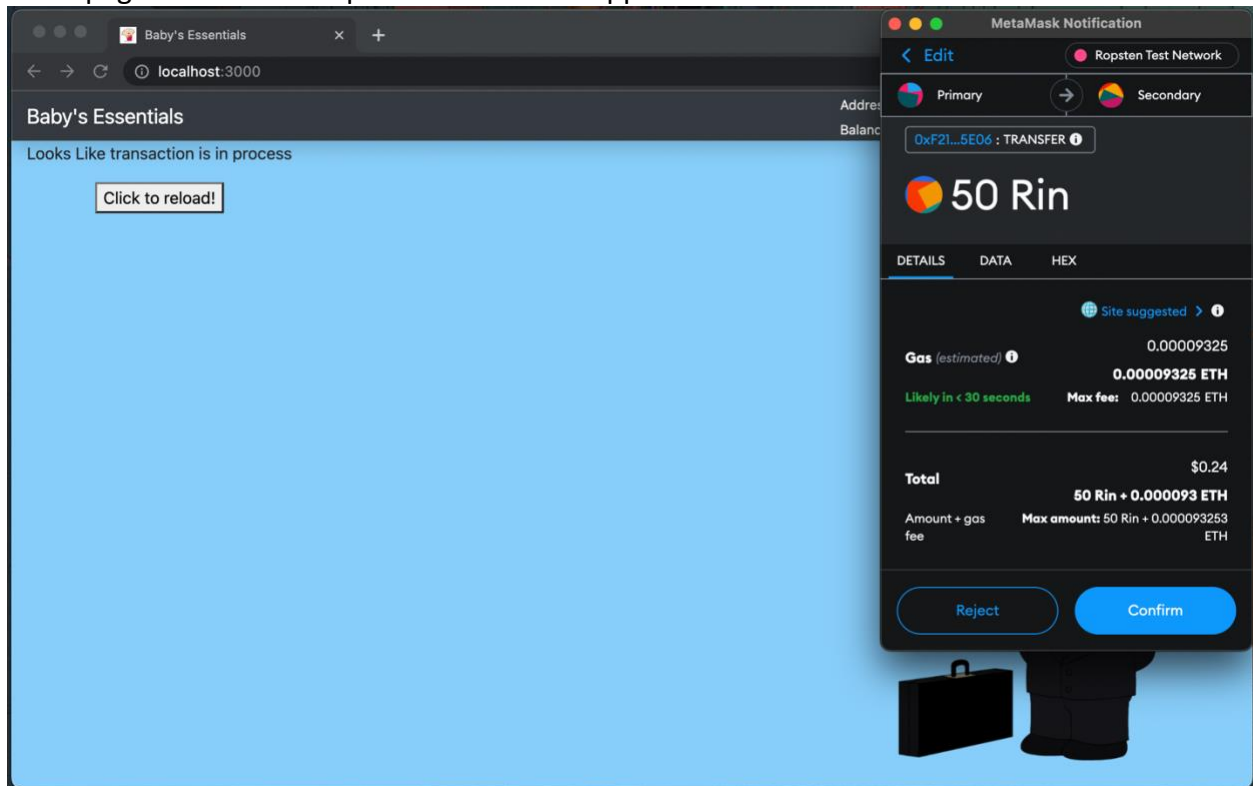
Product Name

Product Price

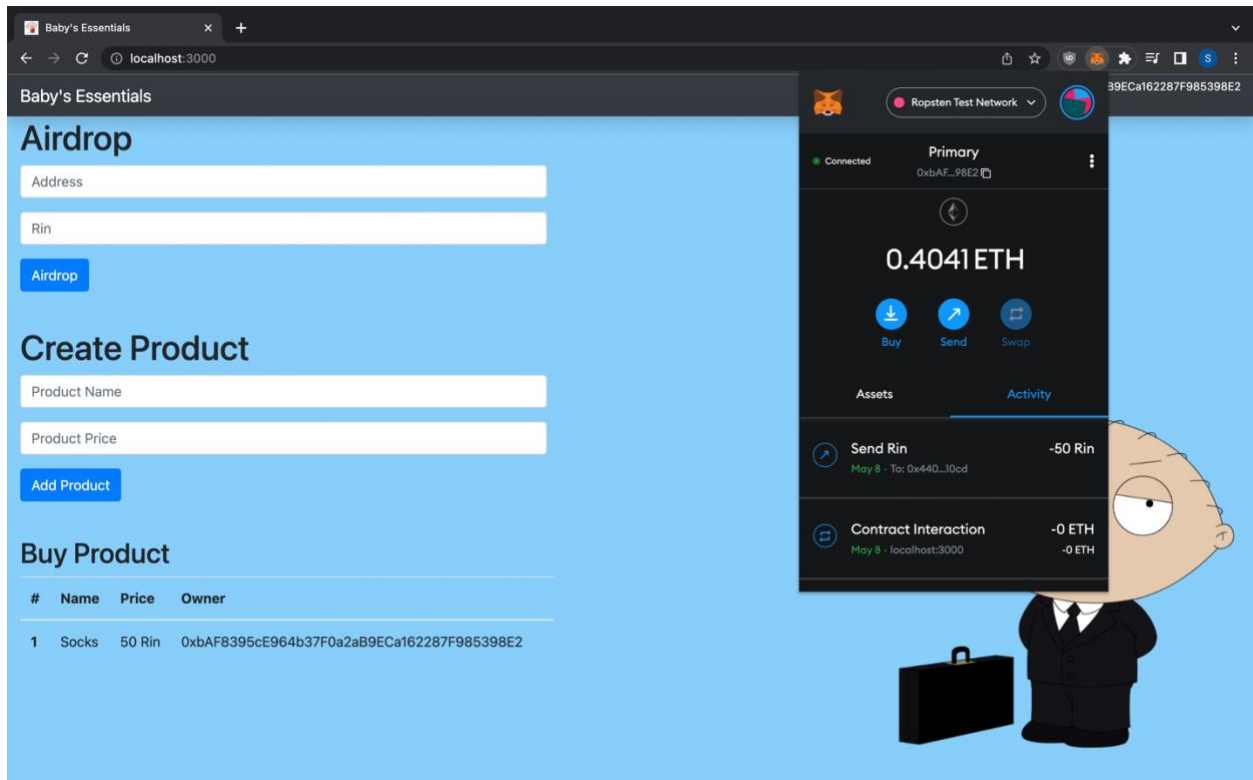
Buy Product

#	Name	Price	Owner
1	Socks	50 Rin	0xbAF8395cE964b37F0a2aB9ECa162287F985398E2

Homepage for our marketplace. All balances appear in **Rin** now.



We airdrop 50 Rins from the primary to the secondary account.



Successfully transferred 50 Rins.

Baby's Essentials

Airdrop

Address

Rin

Airdrop

Create Product

Product Name

Product Price

Add Product

Buy Product

#	Name	Price	Owner
1	Socks	50 Rin	0xbAF8395cE964b37F0a2aB9ECa162287F985398E2
2	Suit	100 Rin	0x440C945e48b703f2E9F1Ece23B3Aa4b86fE410cD

Purchase

Primary

Connected

0.4041 ETH

Buy Send Swap

Assets Activity

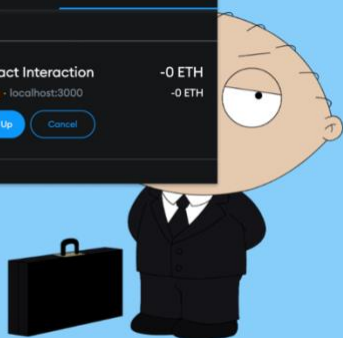
Queue (1)

Contract Interaction

Pending - localhost:3000

Speed Up Cancel

-0 ETH -0 ETH



We will now buy the Suit listed by the secondary account using the Primary account.

Baby's Essentials

Airdrop

Address

Rin

Airdrop

Create Product

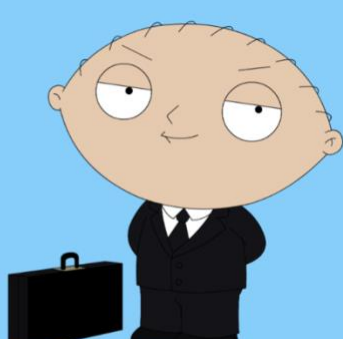
Product Name

Product Price

Add Product

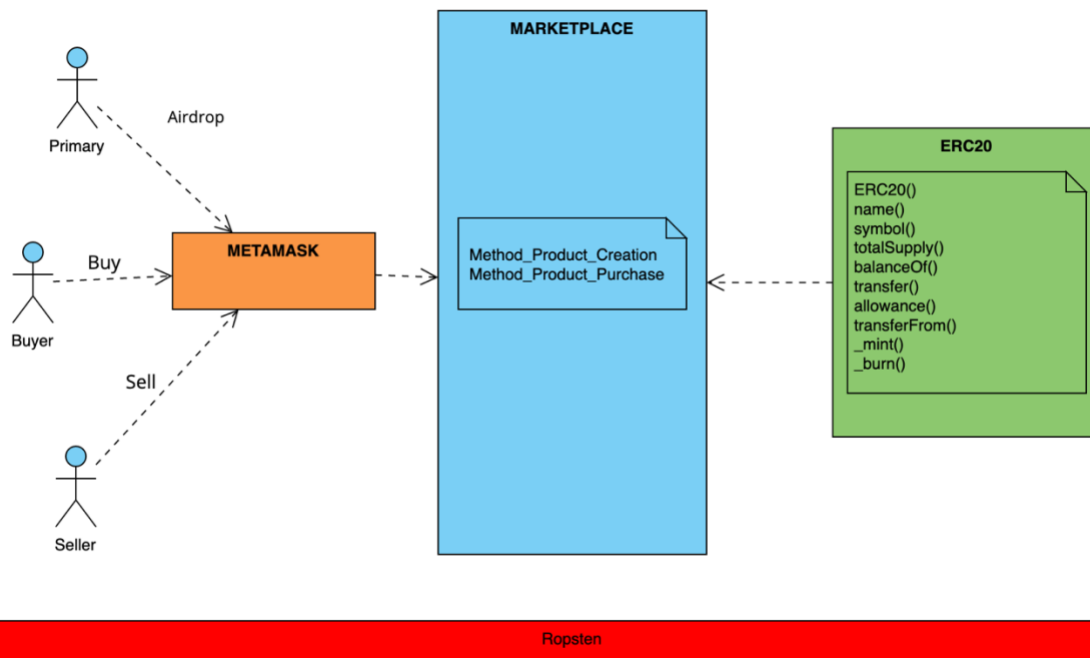
Buy Product

#	Name	Price	Owner
1	Socks	50 Rin	0xbAF8395cE964b37F0a2aB9ECa162287F985398E2
2	Suit	100 Rin	0xbAF8395cE964b37F0a2aB9ECa162287F985398E2



We successfully buy the Suit. Balance on the Primary account goes down by 100 Rin.

8. Structure



Overall structure.

9. Deployment

Deployment on Ropsten is achieved using the commands given below in our app dapp directory.

```
npm install
```

- Installs required modules.

```
npm install --save truffle-hdwallet-provider
```

- Installs hdwallet.

```
truffle deploy --network ropsten
```

- Deploys the app on the ropsten network.

```
npm run start
```

- Starts the dapp.