BITS F452 Blockchain Tech Assignment 3

By:-

Visweswar Sirish Parupudi (2020AAPS0330H)

Konkala Rithvik (2020A8PS0517H)

Gautham Gutta (2020AAPS2204H)

Gokul Pradeep (2018B5A70785H)

<u>Title of the project</u>

Decentralized Application for In-Game transactions

IMPORTANT NOTE :- Please run the following code on VSCode, and make sure that VSCode has the TailwindCSS extension installed, the metamask extension for google chrome is installed, and NodeJS is installed on your system

Overview of the Project:-

In-game transactions add fun to a game. However, there are various security concerns when it comes to online transactions, and hackers usually manage to cheat the protocol in some way or the other.

In this project, we attempt to create a secure platform where players of a game can trade skins and other valuable items, for some amount of in-game currency.

We aim to achieve this by using smart contracts programmed in solidity, wherein the transactions are stored in the ethereum blockchain.

When a user puts an in-game asset for sale, the ownership of the item will be transferred from the creator to the marketplace contract.
When a user purchases that particular asset, the purchase price will be transferred from the buyer to the seller and the item will be transferred from the marketplace to the buyer.

The marketplace owner will be able to set a listing fee. This fee will be taken from the seller and transferred to the contract owner upon completion of any sale, enabling the owner of the marketplace to earn recurring revenue from any sale transacted in the marketplace.
Here, we treat our in-game assets as non-fungible tokens(NFTs) since they are similar in nature

The marketplace logic will consist of just one smart contract, which allows users to mint in-game assets and list them in the marketplace.

Frontend of our marketplace:-

- In the part of the code "**app.js**", we design the interface of our marketplace website in Javascript. We design our homepage such that the navigation has links for the home route as well as a page to make an NFT and then sell it(`<Link href="/create-nft">`), view the NFTs you have purchased(`<Link href="/my-nfts">`), and a dashboard to see the NFTs you've listed(`<Link href="/dashboard">`).
- In the part of the code "**index.js**", we make the main entry-point of the app, and will be the view where we query for the NFTs for sale and render them to the screen. When the page loads, we query the smart contract for any NFTs that are still for sale and render them to the screen along with metadata about the items and a button for purchasing them.
- Next, in the part of the code called "**create-nft.js**"we create the page that allows users to create and list NFTs.

  There are a few things happening in this page:

1. The user is able to upload and save files to IPFS
2. The user is able to create a new NFT
3. The user is able to set metadata and price of item and list it for sale on the marketplace

After the user creates and lists an item, they are re-routed to the main page to view all of the items for sale.

- In the part of the code called "**my-nfts.js**", we will use the `fetchMyNFTs` function that only returns the NFTs owned by the user, which was created in the "NFTMarketplace.sol" smart contract to fetch and render them.
- In the part of the code called "**dashboard.js**", we will be creating is the dashboard that will allow users to view all of the items they have listed.This page will be using the `fetchItemsListed` function from the NFTMarketplace.sol smart contract which returns only the items that match the address of the user making the function call.

__**Backend of our marketplace:-**__

1) **Solidity :**

Solidity is an object-oriented programming language created specifically by the Ethereum Network team for constructing and designing smart contracts on Blockchain platforms.

It's used to create smart contracts that implement business logic and generate a chain of transaction records in the blockchain system.

In our current problem statement, we have to draft a smart contract between 2 players who wish to trade in-game assets. The smart contract we have use is named “**NFTMarketplace.sol”**

We have to make it such that the buyer gets the product they ordered and only after buyer’s acknowledgement the seller gets the money and buyer gets the goods.

We deploy a smartcontract for this scenario with the following logic.

2) STEPS OF TRANSACTION :

STEP 1

The seller has to draft the smart contract for a product.

Then seller pays a security deposit into the smart contract which is double the specified cost of the product they desire to sell

This is to ensure that the seller has an incentive to actually send the promised product to the buyer, in case things go wrong the security deposit is lost and used to make up for the buyer's loss.

STEP 2

Involves the buyer confirming that he wants the product by paying a security deposit (in this case also twice the value of price of requested goods)

The buyer security deposit and seller security deposits are refunded after the transaction.

The refunds are the incentives for the buyer and seller to honestly report the status of delivery of goods.

STEP 3

now the seller ships the promised goods to the buyer.

STEP 4

Buyer now confirms the product on contract and start the process for his refund of security deposit minus the cost of good.

So now the buyer has the goods and half of his security deposit back since the other half is used up as payment for goods automatically by the contract.

STEP 5

Now the seller initiates the payment function for themselves (this function can only be initiated after confirmation from buyer is received).

Seller now gets his security deposit back (twice the value of goods) plus half the security deposit from the buyer (compensation for the goods he sent).

Hence the net transaction of the value of goods is handled by automation and is completely fair for both the involved parties.

3) Variables description :

Seller - is an address type variable that stores the value of the seller (the person who drafts the contract, in this case xyz.com)

Buyer - is an address type variable that stores the value of the buyer (The person who requests goods from the seller)

Cost – is an uint type variable that stores the cost of the goods that are exchanged in the transaction between buyer and seller

State – is enum (user-defined data types) and has the 4 states of transaction which are

Created - contract enters this state when seller deploys the contract.

Locked - contract enters the state when buyer puts in a value equal to twice the cost.

Release - contract enters this state when buyer confirms that he received the product.

Inactive - contract enters this state when seller is refunded and given the extra paid by buyer.

4) Function description :

We create a class named MarketItem, and tokenid, seller, owner, price, and sold status are all declared within this class

We create an event named MarketItemCreated, and tokenid, seller, owner, price, and sold status are all declared within this event

We declare a function function updateListingPrice(uint _listingPrice) which updates the contract price, upon input from the owner

We declare a function “function createToken()” which creates the token, and the inputs of this function are the token price and the token URL.

We declare a function “function createMarketSale()” which initiates the transaction between 2 players

We declare a function “function fetchMarketItems()” which returns the list of in-game assets which are up for sale

We declare a function “function fetchMyNFTs()” which returns the assets which are in possession of the current user account

Now the smart contract code and environment is complete and we can try testing it out.
To do so, we can create a local test to run through much of the functionality, like minting a token, putting it up for sale, selling it to a user, and querying for tokens.

Running the project:-

To run the project, we will need to have a deploy script to deploy the smart contracts to the blockchain network.

Deploying the contracts to a local network

When we created the project, Hardhat created an example deployment script at scripts/deploy.js.

This script will deploy the contract to the blockchain network and create a file named config.js that will hold the address of the smart contract after it's been deployed.

We will first test this on a local network

To spin up a local network, open terminal and run the following command:

```
npx hardhat node
```

This should create a local network with 20 accounts.

Next, keep the node running and open a separate terminal window to deploy the contract.

In a separate window, run the following command:

```
npx hardhat run scripts/deploy.js --network localhost
```

When the deployment is complete, the CLI should print out the address of the contract that was deployed.You should also see the config.js file populated with this smart contract address.

Importing accounts into MetaMask:-

You can import the accounts created by the node into your Metamask wallet to try out in the app.
Each of these accounts is seeded with 10000 ETH.
To import one of these accounts, first switch your MetaMask wallet network to Localhost 8545.
Next, in MetaMask click on Import Account from the accounts menu:
Copy then paste one of the Private Keys logged out by the CLI and click Import.
Once the account is imported, you should see some the Eth in the account:
It’s recommended that the user does this with 2 or 3 accounts so that the user has the ability to test out the various functionality between users.

Running the app:-

Now we can test out the app!
To start the app, run the following command in your CLI:

```
npm run dev
```

To test everything out, try listing an item for sale, then switching to another account and purchasing it.