Assignment -3 : Tokenizing Supply Chain Products using NFTs

Deadline : 11th June 11:59 am (Tuesday).
Github Submission Guidelines:
https://github.com/pclubiitk/Lord-of-Chains-25.git

1) Fork this repository on Github.
2) Clone your fork to your local machine.
3) Work on your assignment in that particular assignment's folder (if it's the first assignment, work in Assignment-3 directory), **create a folder inside the assignments folder** (Naming convention: <your name>-<your roll no.>-<assignment number> Ex: Srijani-231033-3) and commit all changes with clear messages.
4) Push your changes to your fork.
5) Create a pull request from your fork's branch to the original repo.
6) Comments are necessary along with a README.md file with clear instructions on how to use your program.
7) Make sure your code has detailed comments.

Objective:

You are required to build an **NFT-based product tracking system** on the blockchain using **Solidity, ERC-721**, and the tools taught so far. The system will simulate a simplified supply chain with **one user account** acting as multiple roles: seller, warehouse, delivery agent, and buyer.

Each product will be **minted as an NFT**, and its **journey through the supply chain will be recorded on-chain** using time stamped events and state updates. This assignment focuses on using **Remix**, **MetaMask**, and **Pinata** (for metadata).

Guidelines:

1) The contract must simulate the following roles:
    a) Seller
    b) Warehouse
    c) Deliver
    d) Buyer
2) These roles can be simulated using your single Metamask account. They do not have to be separate Ethereum accounts. Instead, your contract should allow the single user to simulate transitions by specifying or selecting the current role.
3) An example to simulate multiple roles for a single account is given below. You may use any other strategy you wish.

```
struct Product {
    uint256 tokenId;
    Role currentRole;
    uint256 createdAt;
    uint256 deliveryDeadline;
}
```

And then keep a mapping:

```
mapping(uint256 => Product) public products;
```

Role-Based Transfer Logic:

Each time you call a function like transferToNextRole(uint256 tokenId), you can require the correct sequence:

```
function transferToNextRole(uint256 tokenId) public {
    Product storage product = products[tokenId];

    if (product.currentRole == Role.Seller) {
        product.currentRole = Role.Warehouse;
    } else if (product.currentRole == Role.Warehouse) {
        product.currentRole = Role.Delivery;
    } else if (product.currentRole == Role.Delivery) {
        product.currentRole = Role.Buyer;
    } else {
        revert("Product already delivered to buyer.");
    }

    // Log the event
    transferLogs[tokenId].push(
        TransferLog({
            fromRole: getRoleName(product.currentRole),  // Previous role
            toRole: getRoleName(product.currentRole),    // Current role
            timestamp: block.timestamp
        })
    );

    emit Transfer(tokenId, product.currentRole, block.timestamp);
}
```

Add a helper function to return role name:

```
function getRoleName(Role role) internal pure returns (string memory) {
    if (role == Role.Seller) return "Seller";
    if (role == Role.Warehouse) return "Warehouse";
    if (role == Role.Delivery) return "Delivery";
    if (role == Role.Buyer) return "Buyer";
    return "Unknown";
}
```

This way, although you are using only one account, the contract logic tracks the current role of the NFT, and you manually simulate role actions step-by-step.

4) NFT Minting
    a) Use **OpenZeppelin's ERC-721** standard. Remember to select required functionalities/features like Mintable, Auto Increment IDs and URI Storage.
    b) When the seller packages a product, mint a new NFT.
5) Since storing detailed metadata **on-chain is costly**, we use off-chain storage and reference it in the NFT using a `tokenURI`. Here's how you can do it step-by-step:

    a) Step 1: Prepare Your Metadata. The metadata should be in JSON format and follow the ERC-721 metadata schema.
    b) Step 2: Upload the metadata to Pinata. This should include Product ID, Name/Description, Timestamp of creation and Delivery Deadline.
    c) Step 3: Use the metadata link in your smart contract.

6) The product must go through the following path:
   `Seller → Warehouse → Delivery → Buyer`
7) Each transfer must be tracked with block.timestamp and emit an event displaying the from-role, to-role and time of transfer.
8) Handling multiple products:
    a) Use ERC-721's built-in incremental token IDs for each product NFT minted.
    b) Store each product's supply chain state and metadata in a mapping:

```
mapping(uint256 => Product) public products;
mapping(uint256 => TransferLog[]) public transferLogs;
```

    c) Mint NFTs dynamically when the seller packages product

```
function mintProductNFT(string memory tokenURI, uint256 deliveryDeadline)
public onlyRole(Role.Seller) returns (uint256) {
    uint256 newTokenId = _tokenIds.current();
    _mint(msg.sender, newTokenId);
    _setTokenURI(newTokenId, tokenURI);

    products[newTokenId] = Product({
        tokenId: newTokenId,
        currentRole: Role.Seller,
        createdAt: block.timestamp,
        deliveryDeadline: deliveryDeadline
    });

    _tokenIds.increment();
    return newTokenId;
}
```

    d) Each product NFT independently tracks its own current role and transfer history.
    e) Role transfer functions take a `tokenId` parameter to operate on specific products.

f) The state and event logs for each NFT allow independent audit and tracking of multiple products on-chain.

9) You **must implement at least 3** of the following **rules** in your contract. You may also implement your own but make sure you explain it to us via comments or README. Add them as modifiers, checks, or additional logic with proper comments:

1. Temperature Sensitivity Rule

Some products require cold storage. For such NFTs, add a boolean flag in metadata or on-chain indicating `isTemperatureSensitive`.

- **Rule:** Warehouse must confirm "cold storage mode" during transfer (simulate via a function call or flag).
- If the warehouse skips this, block the transfer to Delivery with a revert.

2. Expiry Date Rule

Each product NFT must store an expiry timestamp.

- **Rule:** If the product's expiry date is passed at any transfer stage, reject transfer (simulate product spoilage).
- If expired, emit an event `ProductExpired(tokenId, currentTimestamp)`.

3. Maximum Transit Time Rule

Set a maximum allowed time that a product can stay in any role (e.g., warehouse or delivery).

- **Rule:** If the product is held longer than allowed in a role, prevent further transfer until an override function is called by the Seller (simulate emergency intervention).

- Emit `TransitDelay(tokenId, role, delaySeconds)` when delay occurs.

4. Role Transfer Limit Rule

Each role can only hold a product for a maximum number of transfers (for example, the warehouse can only transfer to delivery 5 times, simulating repeated rejections or returns).

- Enforce this limit and revert if exceeded.
- Emit event `TransferLimitExceeded(tokenId, role)`.

5. Audit Log Rule

Require that each transfer includes an audit note (string) that must be recorded on-chain with the transfer event.

- If the audit note is empty, revert the transaction.

.

10) Delivery Time Auditing: Each product must have an expected **delivery deadline** set at minting time.

a) When the product reaches the **Buyer**, compare the current timestamp to the `deliveryDeadline`.
b) Emit one of the following events:
    i) DeliveryStatus(tokenId, "OnTime", timestamp)
    ii) DeliveryStatus(tokenId, "Delayed", timestamp)

11) Transaction History: For every transition (e.g., Seller → Warehouse), the history should capture:

a) From Role: Who was handling the product before the transfer.

b) To Role: Who is handling the product after the transfer.

c) Timestamp: When the transfer happened. This is pulled from `block.timestamp` — the time the blockchain recorded the transaction.

d) Audit Note: A short message explaining the context or reason for the transfer (like “Product checked and packaged”, “Cold storage confirmed”, etc.).