Blockchain Arena

Simulating Mining Wars and Network Attacks

Final Assignment

Verifiable On-Chain Provenance Tracker

Release Date: July 06, 2025

Due Date: 23:59hrs, July 16, 2025

Objective

Welcome to your final assignment! In this project, you will build a complete decentralized application (DApp) to register unique items and track their ownership history on the blockchain. This will create a verifiable, tamper-proof record of provenance, which is the core concept behind Non-Fungible Tokens (NFTs). Think of it as a system for tracking high-value goods like luxury watches, university degrees, or digital art certificates.

Learning Objectives

This project will solidify your understanding of:

- **Smart Contract Development:** Writing, testing, and deploying contracts using Solidity and Remix IDE.
- **Custom Data Structures:** Using `struct` to model real-world objects on-chain.
- **State Management:** Using `mappings` to store and retrieve data efficiently.
- **Event-Driven Architecture:** Using `events` to log important state changes and communicate with the frontend.
- **Access Control:** Writing secure functions and protecting state changes with modifiers or `require` statements.
- **Full-Stack DApp Integration:** Connecting a web frontend to a smart contract backend using the `ethers.js` library.
- **Event Querying:** Reading on-chain event logs to reconstruct item history.

Tools and Environment

- **Smart Contract IDE:** I strongly recommend using the **Remix IDE** (https://remix.ethereum.org/) for writing, compiling, and deploying your Solidity smart contract. It is a web-based IDE that requires no setup.
- **GitHub Integration:** Link your Remix IDE to your GitHub account to securely manage and back up your code. Since Remix is browser-based, connecting to GitHub helps prevent accidental data loss.
- **Test Network:** You will deploy your contract to an Ethereum test network (e.g., **Sepolia**). You can get free test ETH from a public faucet.
- **Frontend:** A simple HTML, CSS, and JavaScript stack.
- **Wallet:** A browser-based wallet like **MetaMask** is required to interact with your DApp.

Core Requirements

This project has two main parts: the smart contract backend and the web frontend.

Part 1: Smart Contract (`Tracker.sol`)

You will write a Solidity smart contract with the following features:

1. **Item Struct:** Create a `struct` named `Item` to store the data for each unique item. It must contain at least:
   - `uint256 id`: A unique identifier for the item.
   - `string name`: A name or description for the item.
   - `address owner`: The Ethereum address of the current owner.
2. **State Variables:**
   - A `mapping` to link a `uint256` ID to its corresponding `Item` struct.
   - A counter variable to ensure every new item gets a unique ID.
3. **Events:** The contract must emit events for key actions to allow the frontend to listen for changes.
   - `event ItemRegistered(uint256 indexed id, address indexed owner, string name);`
   - `event OwnershipTransferred(uint256 indexed id, address indexed from, address indexed to);`
4. **Functions:**
   - `function registerItem(string memory _name)`: Creates a new item with a unique ID, sets the function caller (`msg.sender`) as the initial owner, and emits an `ItemRegistered` event.
   - `function transferOwnership(uint256 _id, address _newOwner)`: Must verify that the `msg.sender` is the current owner of the item before updating the owner and emitting an `OwnershipTransferred` event.

Part 2: Frontend (HTML, CSS, JavaScript with `ethers.js`)

Build a simple web interface to interact with your deployed smart contract.

1. **Wallet Connection:** A button to allow users to connect their MetaMask wallet to your DApp.
2. **Register New Item:** A form with a text input for the item's name and a "Register" button that calls the `registerItem` function.
3. **Display Items:** List all registered items. The best way to do this is by fetching and listening for the `ItemRegistered` events.
4. **Transfer Ownership:** For each item they own, the connected user should see an input field and a "Transfer" button to assign a new owner by calling the `transferOwnership` function.
5. **View Provenance:** Allow a user to click on any item to view its complete ownership history. This will require you to fetch all `OwnershipTransferred` events related to that specific item's ID.

Bonus Features (Optional)

If you finish the core requirements early, try implementing one of these:

- **Enhanced Metadata:** Add more fields to your `Item` struct, like a URL to an image or a longer description.
- **Burnable Items:** Implement a `burnItem(uint256 _id)` function that allows an owner to permanently destroy their item's record (be sure to add proper access control).
- **Approval Function:** Add an `approve(address _to, uint256 _id)` function, similar to the ERC-721 standard, that allows an owner to grant another address permission to transfer the item on their behalf.

Submission Guidelines

Submit the GitHub repository link containing the following files:

1. Source code for your smart contract (`Tracker.sol`).
2. All frontend source code files (HTML, CSS, JS).
3. A `README.md` file with:
    - Instructions for running the frontend.
    - The deployed contract address on the Sepolia test network.
    - A link to the contract on a block explorer like Etherscan.

Submission Link: Google Form for Submission
Deadline: 23:59hrs, July 16, 2025.

GOOD LUCK & ENJOY BUILDING!