**Why URI generation and blockchain storage is preferred over off-blockchain storage of the URI data:**

**Blockchain Storage**:

* The URI is stored directly on the blockchain as part of the ERC-721 metadata. It means **the blockchain will hold the actual ownership record of that NFT** and the associated information (in the form of the URI).
* This makes the data verifiable and tamper-proof, since the blockchain's immutability ensures the URI can't be altered once the NFT is minted.
* // contracts/GameItem.sol
* // SPDX-License-Identifier: MIT
* pragma solidity ^0.8.0;
* import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
* import "@openzeppelin/contracts/utils/Counters.sol";
* contract GameItem is ERC721URIStorage {
* using Counters for Counters.Counter;
* Counters.Counter private \_tokenIds;
* constructor() ERC721("GameItem", "ITM") {}
* // 1. Hash Verification Process:
* //seller hashes the contract details:
* function hash\_seller(string memory contract\_details) public pure returns(bytes32){
* return keccak256(abi.encodePacked(contract\_details));
* }
* //buyer hashes the contract details received from the seller:
* function hash\_buyer(string memory contract\_details) public pure returns(bytes32){
* return keccak256(abi.encodePacked(contract\_details));
* }
* //buyer verifies the hashes match:
* function verify\_hashCont (bytes32 buyer\_hash, bytes32 seller\_hash) public pure returns (bool){
* if (buyer\_hash == seller\_hash){
* return true;
* } else {
* return false;
* }
* }
* // 2. Payment verification process:
* //buyer hash the payment details:
* function hash\_paymentB (string memory name\_on\_card, int256 amount) public pure returns (bytes32){
* return keccak256(abi.encodePacked(name\_on\_card, amount));
* }
* //seller receives the hash and the payment details, this verifies the transaction:
* function hash\_paymentS (string memory name\_on\_card, int256 amount) public pure returns (bytes32){
* return keccak256(abi.encodePacked(name\_on\_card, amount));
* }
* //seller verifies the hashes match and manually aknowledges the receipt of payment:
* function verify\_hashPayment (bytes32 pay\_hashB, bytes32 pay\_hashS, string memory aknowledegement\_of\_payment) public pure returns(bool, bytes32){
* if (pay\_hashB == pay\_hashS && keccak256(abi.encodePacked(aknowledegement\_of\_payment)) == keccak256(abi.encodePacked("Payment Received"))) {
* bytes32 combined\_hash = keccak256(abi.encodePacked(pay\_hashS, pay\_hashB, aknowledegement\_of\_payment));
* return (true, combined\_hash);
* } else {
* return (false, bytes32(0));
* }
* }
* // 3. Generate NFT
* //generate URI
* ///
* // Function to convert bytes32 to a hex string
* function bytes32ToHexString(bytes32 data) public pure returns (string memory) {
* bytes memory alphabet = "0123456789abcdef";
* bytes memory str = new bytes(64); // A bytes32 value will result in a 64-character hex string
* for (uint256 i = 0; i < 32; i++) {
* str[2\*i] = alphabet[uint8(data[i] >> 4)]; // Get the first 4 bits
* str[2\*i+1] = alphabet[uint8(data[i] & 0x0f)]; // Get the last 4 bits
* }
* return string(str);
* }
* ///(ChatGPT, 2023)
* // define function that generates the URI if the hashes are verified:
* function generateURI(bytes32 cont\_ver\_hash, bytes32 pay\_ver\_hash) public pure returns (string memory) {
* //convert to hexadecimal string
* string memory cont\_hash\_hex = bytes32ToHexString(cont\_ver\_hash);
* string memory pay\_hash\_hex = bytes32ToHexString(pay\_ver\_hash);
* //generate URI
* string memory uri = string(abi.encodePacked("https://Store\_Scontracts.com/contractHash=", cont\_hash\_hex,"&paymentHash=", pay\_hash\_hex));
* return uri;
* }
* // contract\_ver\_Hash will have to be provided by the buyer
* // payment\_ver\_Hash will have to be provided by the seller
* //generate the NFT only if the hashes are verified:
* //the NFT function will be on the seller side, but the seller will only be able to provide the payment hash.
* function awardItem(address buyer\_wallet\_address, bytes32 cont\_ver\_hash, bytes32 pay\_ver\_hash,
* string memory aknowledegement\_of\_payment) public returns (uint){
* ///
* require(verify\_hashCont(cont\_ver\_hash, cont\_ver\_hash), "Contract hash not verified");
* require(verify\_hashPayment(pay\_ver\_hash, pay\_ver\_hash, aknowledegement\_of\_payment), "Payment hash not verified");
* /// (ChatGPT, 2023)
* //generate the URI if true
* string memory uri = generateURI(cont\_ver\_hash, pay\_ver\_hash);
* //mint the NFT
* uint256 newItemId = \_tokenIds.current();
* \_mint(buyer\_wallet\_address, newItemId);
* \_setTokenURI(newItemId, uri);
* \_tokenIds.increment();
* //token ID of the new NFT
* return newItemId;
* }
* }