Flipkart GRiD 4.0

2022 Campus Challenge

Blockchain-based eCommerce warranty system using NFTs

Team Name: Vcodeorganic

Institute Name: Indian Institute of Technology (IIT), Jodhpur

/TEAM_MEMBER_DETAILS

TEAM VCODEORGANIC

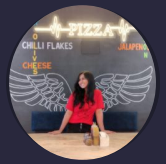
Indian Institute of Technology (IIT), JODHPUR

/CHINMAY BORALE

Team Leader
Batch of 2023

/GAURAV SEN

Team Member
Batch of 2023



/SAUMYA VAISH

Team Member
Batch of 2023

/DELIVERABLES FOR LEVEL 2

- The blockchain smart contract allows users to prove ownership
- Provides the purchasing history, warranty period, and other item information
- The warranty card includes the item’s serial number and upon purchase be sent to the customer’s smartphone.
- The NFTs are decaying in nature, in that, after a certain period their use for the redemption of warranty benefits offered by the brand/retailer will expire

/USE CASES

P01 > Tracing ownership chain

P02 > Control number of warranty transfers

P03 > Combating counterfeits

P04 > NFTs cannot be falsified

/SOLUTION APPROACH

/PRODUCT SALE

Company mails all files to the user for minting the NFT

/MINT NFT

User mints the NFT using his wallet address

/METADATA

NFT's metadata contains all the information regarding the warranty

/AVAIL WARRANTY

User proves ownership of NFT to avail warranty benefits

/FUNCTIONS

/_mint(p1,p2)

PARAMS

address to
uint256 tokenId

/transferOwnership(p1)

PARAMS

address newOwner

/ownerOf(p1)

PARAMS

uint256 tokenId

/_burn(p1)

PARAMS

uint256 tokenId

/LIMITATIONS

- Users need to mint the NFTs themselves hence blockchain experience is a prerequisite
- Currently, all commands need to be input at remix IDE manually instead of direct commands

/FUTURE SCOPE

- Build a GUI so that the service can be availed by users easily
- Attach a barcode scanner along with product to access the GUI directly
- Introduce loyalty program for more user engagement

NFT METADATA JSON FILE TEMPLATE

The core of an NFT which contains information about its name, description and traits.

The metadata will be input to the smart contract which is deployed on the Ethereum Network.

```
{
  "name": "product_name",
  "description": "product_description",
  "image": "image_url",
  "attributes": [
    {
      "trait_type": "Serial_number",
      "value": "serial_number"
    },
    {
      "trait_type": "Warranty_period",
      "value": "x_years"
    },
    {
      "trait_type": "Owner",
      "value": "owner_wallet_address"
    },
    {
      "trait_type": "Benefits",
      "value": "benefits_description"
    },
    {
      "trait_type": "Terms_and_conditions",
      "value": "terms_and_conditions_description"
    }
  ]
}
```

```
/**
 * @dev Mints `tokenId` and transfers it to `to`.
 *
 * WARNING: Usage of this method is discouraged, use {_safeMint} whenever possible
 *
 * Requirements:
 *
 * - `tokenId` must not exist.
 * - `to` cannot be the zero address.
 *
 * Emits a {Transfer} event.
 */
function _mint(address to, uint256 tokenId) internal virtual {
    require(to != address(0), "ERC721: mint to the zero address");
    require(!_exists(tokenId), "ERC721: token already minted");

    _beforeTokenTransfer(address(0), to, tokenId);

    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(address(0), to, tokenId);
}
```

MINTING AN NFT

TRANSFERRING THE OWNERSHIP

```
/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _setOwner(newOwner);
}
```

```
/**
 * @dev Returns the owner of the `tokenId` token.
 *
 * Requirements:
 *
 * - `tokenId` must exist.
 */
function ownerOf(uint256 tokenId) public view virtual override returns (address) {
    address owner = _owners[tokenId];
    require(owner != address(0), "ERC721: owner query for nonexistent token");
    return owner;
}
```

VERIFYING THE OWNERSHIP

BURN NFTs

```
/**
 * @dev Destroys `tokenId`.
 * The approval is cleared when the token is burned.
 *
 * Requirements:
 *
 * - `tokenId` must exist.
 *
 * Emits a {Transfer} event.
 */
function _burn(uint256 tokenId) internal virtual {
    address owner = ERC721.ownerOf(tokenId);

    _beforeTokenTransfer(owner, address(0), tokenId);

    // Clear approvals
    _approve(address(0), tokenId);

    _balances[owner] -= 1;
    delete _owners[tokenId];

    emit Transfer(owner, address(0), tokenId);
}
```