BlockCommerce

Document

I. Giới thiệu

BlockCommerce cung cấp một bộ sản phẩm bảo mật bằng blockchain với smart contract để xây dựng, quản lý và kiểm tra tất cả các khía cạnh của hoạt động và phát triển phần mềm cho các dự án sàn thương mại trên Ethereum.

II. Sản phẩm cung cấp

Contract : Một thư viện các hợp đồng thông minh bao gồm hợp đồng tạo, lưu trữ NFT và hợp đồng đóng vai trò như một thị trường cho viện buôn bán trao đổi NFT. Các hợp đồng này có thể tái sử dụng, bảo mật và được lưu trữ trên blockchain, được viết bằng Solidity. Bên cạnh đó, các chức năng trong function được phân quyền theo tài khoản có thể gọi.

Proxy: Một bộ hợp đồng và công cụ để triển khai và quản lý các hợp đồng, nó cho phép tách các hợp đồng có thể nâng cấp trên Ethereum.

Lazy minting: Chức năng nâng cao, giúp người bán không phải chịu bất kì chi phí nào từ lúc tạo một sản phẩm đến lúc sáng phẩm đó được bán đi. Quy phí về cho người dùng cuối chịu trách nhiệm

Xác thực: Chức năng nâng cao, giúp người dùng, các bên mau bán có thể tự kiểm soát hợp đồng và thực hiện hợp đồng theo mong muốn, thay cho việc thực hiện tự động vốn có của hợp đồng.

1. Contract:

Art

Chuẩn ERC721 là một tiêu chuẩn để đại diện cho quyền sở hữu của các mã thông báo không thể thay thế, nghĩa là mỗi mã thông báo là duy nhất, với nhiều phần mở rộng tùy chọn và được chia thành nhiều hợp đồng. Hợp đồng cung cấp sự linh hoạt về cách kết hợp các hợp đồng này, cùng với các tiện ích mở rộng hữu ích tùy chỉnh.

Hợp đồng Art được xây dựng và kế thừa dựa trên chuẩn ERC721, đảm bảo các NFT được tạo là duy nhất với tokenID được tự động tạo theo thứ tự, chỉ định vị trí lưu (baseURL) trên IPFS. Đối với các hợp đồng kế thừa từ hợp đồng Art, muốn thay đổi thành một vị trí lưu trữ khác có thể override lại hàm _baseURL được chỉ định chế độ hiển thị internal và đặt vào đó URL nơi lưu trữ cho NFT của bạn.

Tích hợp:

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.0;

import “@blockcommerce/contracts/Art.sol”
```

```
contract YourContractName is Art{
    /*
        …Your state
    */
    /*
        …Your event
    */
    /*
        …Your modifier
    */
    constructor (string memory name, string memory symbol) Art (name, symbol){}
    /*
        …Your new function or logic
    */
}
```

Các hàm:

createNewNFT(_to, string _hash)

Tạo một NFT với địa chỉ của người sẽ nhận _to được từ mã băm _hash của sản tác phẩm nghệ thuật

setOwnerToTokenId(_to, _tokenId)

Thiết lập lại quyền owner cho một địa chỉ _to với mã token _tokenId muốn chuyển, hàm này chỉ được gọi bởi người sở hữu của _tokenId.

getTokenList(_owner)

Trả về danh sách các NFT của một địa chỉ _ owner

balanceOf(owner)

Trả về số lượng NFT mà địa chỉ owner đang sở hữu

ownerOf(tokenId)

Trả về địa chỉ người sở hữu của toekn có mã tokenId

transferFrom(from, to, tokenId)

Người chủ hoặc người được ủy quyền sẽ có quyền gọi hàm này với mục đích chuyển một token có mã tokenId từ chủ sở hữu có địa chỉ from tới người sở hữu mới có địa chỉ to.

approve(to, tokenId)

Chủ sở hữu của token có mã tokenId sẽ có quyền ủy quyền cho một địa chỉ to nào đó sử dụng token của mình. Có quyền sử dụng không đồng nghĩa với việc sở hữu.

getApproved(tokenId)

Lấy ra danh sách những người được ủy quyền sử dụng tokenId

setApprovalForAll(operator, _approved)

Chủ sở hữu thiết lập quyền sử dụng hoặc thu hồi quyên sử dụng tất cả các token của mình cho địa chỉ operator với _ approved là true hoặc false.

isApprovedForAll(owner, operator)

Trả về true hoặc false để xác nhận operator có quyền sử dụng tất cả các token của địa chỉ owner hay không.

_baseURI()

Dùng để thiết lập base URL, nơi bạn sẽ lưu thông tin metadata cho NFT bạn sẽ tạo.

_mint()

Hàm internal dùng để đào một NFT tới địa chỉ của người gọi

ImarketPlace

Tích hợp:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;
import “@blockcommerce/ contracts /intefaces /ImarketPlace.sol”
contract YourContractName {
    ImarketPlace public markerPlace;
    /*
        …Your state
    */
    /*
        …Your event
```

```
        */
        /*
            …Your modifier
        */
        constructor (address _deployedAddr) {
            marketPlace = ImarketPlace(_deployedAddr);
        }
        /*
            …Your new function or logic
        */
        function example (string memory_hashInfo, string memory _hashImg, uint256
_price) public onlyOwner{
            marketPlace.createNewProduct(_hashInfo, _hashImg,  _price);
            /* other logic */
        }
    }

    // SPDX-License-Identifier: UNLICENSED
    pragma solidity ^0.8.0;
    import "@blockcommerce/ contracts intefaces/ImarketPlace.sol"
    contract YourContractName is ImarketPlace {
        /*
            …Your state
        */
        /*
            …Your event
        */
        /*
            …Your modifier
        */
```

```
        /*
            …Your new function or logic
        */
        function createNewProduct (string memory_hashInfo, string memory
_hashImg, uint256  _price) public override onlyOwner returns (uint256){
            /* other logic */
        }
    }
```

Các hàm:

setArtAddr(address _artAddr)

Thiết lập địa chỉ của contract Arrt đã được triển khai lên mạng Ethereum

createNewProduct(_hashInfo, _hashImg, _price)

Tạo một sản phẩm mới với thông tin đã được băm _hashInfo, hình ảnh đã được băm _hashImg, và mức giá _price muốn bán cho sản phẩm này

setListOrNot(_tokenId)

Thiết lập cho một sản phẩm với _tokenId có được liệt kê hiển thị trên sàn hay không

setSellOrNot(_tokenId)

Thiết lập cho một sản phẩm với _tokenId có đang được bán trên sàn hay không

setPrice(_tokenId, _price)

Dùng để thiết lập giá cho một sản với _tokenId

getProductListCreated(_user)

Lấy ra danh sách tất cả sản phẩm mà địa chỉ _user đã tạo

buyWithETH(_tokenId)

Chức năng này dùng để mua một sản phẩm _tokenId đang được đăng bán trên sàn, và được thanh toán bằng Ether (ETH). Các thiết lập liên quan sẽ được tự động thực hiện.

buyWithCurrency(_tokenId, traransactionId)

Chức năng này dùng để thiết lập lại quyền sở hữu đối với một sản phẩm _tokenID đang được đăng bán trên sàn và lưu lại với giao dịch traransactionId bằng tiền tệ thực hiện thành công.

getProducListOwnable(_owner)

Lấy ra danh sách các sản phẩm thuộc sở hữu của địa chỉ _owner.

offer(_tokenId, _amount, _token20, _timeout)

Tạo yêu cầu để thanh toán một sản phẩm _tokenId với sô lượng _amount token ERC20 _token20 mong muốn, với thời gian tồn tại cho một yêu cầu hợp lệ

restartOffer(_tokenId, _index, timeout)

Khi một yêu cầu hết hạn, người tạo ra yêu cầu có thời gia hạn thêm thời gian cho yêu cầu đó

getOffer(_tokenId, _index)

Lấy ra yêu cầu của sản phẩm _tokenId với vị trí index truyền vào. Dùng trong thanh toán khi yêu cầu được chấp nhận.

approveOffer(_tokenId, _index)

Chủ sở hữu của _tokenId tức là người bán sẽ gọi hàm này khi họ chấp nhận một yêu cầu với index được truyền vào

Event:

Transfer(_tokenId, _oldOwner, _newOwner)

Phát ra một sự kiện thông báo rằng sản phẩm _tokenId đã được chuyển thành công từ địa chỉ chủ sở hữu cũ _oldOwner đến chủ sở hữu mới _newOwner

NewOffer(_tokenId, _amount, _token20, _bargainer, _timeout)

Phát ra một sự kiện thông báo rằng người trả giá đã tạo thành công yêu cầu đối với một sản phẩm _tokenId

ApproveOffer(_tokenId, _oldOwner, _newOwner, _index)

Phát ra một sự kiến thông báo người bán đã chấp nhận yêu cầu, và yêu cầu đã được thực hiện thành công.

2. Proxy

Tại sao cần phải upgrade smart contract?

Một smart contract một khi đã được deploy lên network thì nó sẽ tồn tại vĩnh viễn trên đó và không thể nào sửa đổi được. Đây chính ưu điểm lớn nhất của Blockchain, tuy nhiên, trong một số trường hợp nó lại là hạn chế.

Chẳng hạn như một lập trình viên phát hiện ra có lỗ hổng trong logic smart contract của mình, nó có thể bị hacker khai thác và gây thiệt hại cho anh ta, bây giờ họ muốn tìm cách làm thế nào để vừa có thể vá được lỗ hổng vừa giữ được toàn bộ dữ liệu của smart contract đó.

Theo tư duy thông thường, lập trình viên sẽ backup hết dữ liệu của smart contract cũ về một nơi nào đó ngoài network, deploy một smart contract mới đã vá lỗ hổng, sau đó migrate lại dữ liệu đã backup lên nó, bởi vì đây là môi trường phi tập trung, việc nâng cấp hoàn toàn khác với hệ thống tập trung, và cách này có những hạn chế sau,:

- Việc này chỉ khả thi khi dữ liệu tương đối ít
- Bắt buộc smart contract mới phải có function để gọi migrate dữ liệu => gây rối cho smart contract
- Tốn nhiều gas, hoặc tốn nhiều transaction, thời gian thực hiện

Như vậy, làm thế nào để một smart contract đã được deploy network, sau khi chạy một thời gian đã lưu rất nhiều dữ liệu trên đó, nếu đột nhiên phát hiện có lỗ hổng trong logic thì vẫn có thể cho smart contract chạy theo một logic khác an toàn hơn mà không cần deploy lại một bản mới rồi migrate dữ liệu?

Về cơ bản, chúng ta sẽ tách hợp đồng ban đầu thành 2 hợp đồng, 1 hợp đồng đóng vai trò như nơi lưu dữ liệu, một hợp đồng là nơi chứa logic, khi cần nâng cấp chỉ việc thay đổi địa chỉ của hợp đồng logic mà vẫn đảm bảo hợp đồng lưu dữ liệu không bị ảnh hưởng.

Hãy liên hệ chúng tôi nếu như bạn quan tâm tới tính năng này. Chúng tôi sẽ cung cấp nhân sự hỗ trợ bạn cùng tài liệu thực hiện nếu như bạn mong muốn.

3. Lazy minting

4. Xác thực