BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

MURASAKI7

Project: Murasaki7NFT - ERC721

Website: https://murasaki7.com/

BlockSAFU Score:

85

Contract Address:

0xefa466c7AD390295969549c941547731d7476d4a

DISCLAMER

BlockSAFU has completed this report to provide a summary of the Smart Contract functions, and any security, dependency, or cybersecurity vulnerabilities. This is often a constrained report on our discoveries based on our investigation and understanding of the current programming versions as of this report's date. To understand the full scope of our analysis, it is vital for you to at the date of this report. To understand the full scope of our analysis, you need to review the complete report. Although we have done our best in conducting our investigation and creating this report, it is vital to note that you should not depend on this report and cannot make any claim against BlockSAFU or its Subsidiaries and Team members on the premise of what has or has not been included in the report. Please remember to conduct your independent examinations before making any investment choices. We do not provide investment advice or in any way claim to determine if the project will be successful or not.

By perusing this report or any portion of it, you concur to the terms of this disclaimer. In the unlikely situation where you do not concur to the terms, you should immediately terminate reading this report, and erase and discard any duplicates of this report downloaded and/or printed by you. This report is given for data purposes as it were and on a non-reliance premise and does not constitute speculation counsel. No one should have any right to depend on the report or its substance, and BlockSAFU and its members (including holding companies, shareholders, backups, representatives, chiefs, officers, and other agents) BlockSAFU and its subsidiaries owe no obligation of care towards you or any other person, nor does BlockSAFU make any guarantee or representation to any individual on the precision or completeness of the report.

ABOUT THE AUDITOR:

BlockSAFU (BSAFU) is an Anti-Scam Token Utility that reviews Smart Contracts and Token information to Identify Rug Pull and Honey Pot scamming activity. BlockSAFUs Development Team consists of several Smart Contract creators, Auditors Developers, and Blockchain experts. BlockSAFU provides solutions, prevents, and hunts down scammers. BSAFU is a utility token with features Audit, KYC, Token Generators, and Bounty Scammers. It will enrich the crypto ecosystem.

OVERVIEW

Mint Function

- No mint functions.

Fees

- NO FEE

Tx Amount

- Owner cannot set max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner cannot blacklist.

Ownership

- Owner cannot take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner cannot limit the number of wallet holdings.

Trading Cooldown

- Owner cannot set the selling time interval.

SMART CONTRACT REVIEW

Token Name	**Murasaki7NFT**
Contract Name	Murasaki7NFT
Contract Address	0xefa466c7AD390295969549c941547731d7476d4a
Deployer Address	0xC8242961D2df3Fd2f5572630aB60c933253924A7
Owner Address	0xC8242961D2df3Fd2f5572630aB60c933253924A7
Gas Used for Buy	*will be updated after the DEX listing*
Gas Used for Sell	*will be updated after the DEX listing*
Contract Created	Sep-09-2022 08:00:33 AM +UTC
Initial Liquidity	*will be updated after the DEX listing*
Liquidity Status	Locked
Unlocked Date	*will be updated after the DEX listing*
Verified CA	Yes
Compiler	v0.8.4+commit.c7e474f2
Optimization	Yes with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

Team Review

The Murasaki7NFT team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 472 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://murasaki7.com/

Telegram Group: https://t.me/murasaki7official

Twitter: https://twitter.com/murasaki7offcl

MANUAL CODE REVIEW

Minor-risk

0 minor-risk code issue found

Could be fixed, and will not bring problems.

Medium-risk

0 medium-risk code issues found

Should be fixed, could bring problems.

High-Risk

0 high-risk code issues found

Must be fixed, and will bring problem.

Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. ERC721

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0)
(token/ERC721/ERC721.sol)

pragma solidity ^0.8.0;

import "./IERC721.sol";
import "./IERC721Receiver.sol";
import "./extensions/IERC721Metadata.sol";
import "../../utils/Address.sol";
import "../../utils/Context.sol";
import "../../utils/Strings.sol";
import "../../utils/introspection/ERC165.sol";

/**
 * @dev Implementation of
https://eips.ethereum.org/EIPS/eip-721[ERC721] Non-Fungible Token
Standard, including
 * the Metadata extension, but not including the Enumerable
extension, which is available separately as
 * {ERC721Enumerable}.
 */
contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {
    using Address for address;
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    // Mapping from token ID to owner address
    mapping(uint256 => address) private _owners;

    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
```

```
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private
_operatorApprovals;

    /**
     * @dev Initializes the contract by setting a `name` and a
`symbol` to the token collection.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view
virtual override(ERC165, IERC165) returns (bool) {
        return
            interfaceId == type(IERC721).interfaceId ||
            interfaceId == type(IERC721Metadata).interfaceId ||
            super.supportsInterface(interfaceId);
    }

    /**
     * @dev See {IERC721-balanceOf}.
     */
    function balanceOf(address owner) public view virtual override
returns (uint256) {
        require(owner != address(0), "ERC721: address zero is not
a valid owner");
        return _balances[owner];
    }

    /**
     * @dev See {IERC721-ownerOf}.
     */
    function ownerOf(uint256 tokenId) public view virtual override
returns (address) {
        address owner = _owners[tokenId];
```

```
        require(owner != address(0), "ERC721: invalid token ID");
        return owner;
    }

    /**
     * @dev See {IERC721Metadata-name}.
     */
    function name() public view virtual override returns (string
memory) {
        return _name;
    }

    /**
     * @dev See {IERC721Metadata-symbol}.
     */
    function symbol() public view virtual override returns (string
memory) {
        return _symbol;
    }

    /**
     * @dev See {IERC721Metadata-tokenURI}.
     */
    function tokenURI(uint256 tokenId) public view virtual
override returns (string memory) {
        _requireMinted(tokenId);

        string memory baseURI = _baseURI();
        return bytes(baseURI).length > 0 ?
string(abi.encodePacked(baseURI, tokenId.toString())) : "";
    }

    /**
     * @dev Base URI for computing {tokenURI}. If set, the
resulting URI for each
     * token will be the concatenation of the `baseURI` and the
`tokenId`. Empty
     * by default, can be overridden in child contracts.
     */
    function _baseURI() internal view virtual returns (string
memory) {
        return "";
```

```
    }

    /**
     * @dev See {IERC721-approve}.
     */
    function approve(address to, uint256 tokenId) public virtual
override {
        address owner = ERC721.ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(
            _msgSender() == owner || isApprovedForAll(owner,
_msgSender()),
            "ERC721: approve caller is not token owner nor
approved for all"
        );

        _approve(to, tokenId);
    }

    /**
     * @dev See {IERC721-getApproved}.
     */
    function getApproved(uint256 tokenId) public view virtual
override returns (address) {
        _requireMinted(tokenId);

        return _tokenApprovals[tokenId];
    }

    /**
     * @dev See {IERC721-setApprovalForAll}.
     */
    function setApprovalForAll(address operator, bool approved)
public virtual override {
        _setApprovalForAll(_msgSender(), operator, approved);
    }

    /**
     * @dev See {IERC721-isApprovedForAll}.
     */
    function isApprovedForAll(address owner, address operator)
```

```
public view virtual override returns (bool) {
        return _operatorApprovals[owner][operator];
    }

    /**
     * @dev See {IERC721-transferFrom}.
     */
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public virtual override {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId),
"ERC721: caller is not token owner nor approved");

        _transfer(from, to, tokenId);
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public virtual override {
        safeTransferFrom(from, to, tokenId, "");
    }

    /**
     * @dev See {IERC721-safeTransferFrom}.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes memory data
    ) public virtual override {
        require(_isApprovedOrOwner(_msgSender(), tokenId),
"ERC721: caller is not token owner nor approved");
        _safeTransfer(from, to, tokenId, data);
```

```
    }

    /**
     * @dev Safely transfers `tokenId` token from `from` to `to`,
checking first that contract recipients
     * are aware of the ERC721 protocol to prevent tokens from
being forever locked.
     *
     * `data` is additional data, it has no specified format and
it is sent in call to `to`.
     *
     * This internal function is equivalent to {safeTransferFrom},
and can be used to e.g.
     * implement alternative mechanisms to perform token transfer,
such as signature-based.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by `from`.
     * - If `to` refers to a smart contract, it must implement
{IERC721Receiver-onERC721Received}, which is called upon a safe
transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeTransfer(
        address from,
        address to,
        uint256 tokenId,
        bytes memory data
    ) internal virtual {
        _transfer(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, data),
"ERC721: transfer to non ERC721Receiver implementer");
    }

    /**
     * @dev Returns whether `tokenId` exists.
     *
     * Tokens can be managed by their owner or approved accounts
```

```
via {approve} or {setApprovalForAll}.
     *
     * Tokens start existing when they are minted (`_mint`),
     * and stop existing when they are burned (`_burn`).
     */
    function _exists(uint256 tokenId) internal view virtual
returns (bool) {
        return _owners[tokenId] != address(0);
    }

    /**
     * @dev Returns whether `spender` is allowed to manage
`tokenId`.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function _isApprovedOrOwner(address spender, uint256 tokenId)
internal view virtual returns (bool) {
        address owner = ERC721.ownerOf(tokenId);
        return (spender == owner || isApprovedForAll(owner,
spender) || getApproved(tokenId) == spender);
    }

    /**
     * @dev Safely mints `tokenId` and transfers it to `to`.
     *
     * Requirements:
     *
     * - `tokenId` must not exist.
     * - If `to` refers to a smart contract, it must implement
{IERC721Receiver-onERC721Received}, which is called upon a safe
transfer.
     *
     * Emits a {Transfer} event.
     */
    function _safeMint(address to, uint256 tokenId) internal
virtual {
        _safeMint(to, tokenId, "");
    }
```

```
    /**
     * @dev Same as
{xref-ERC721-_safeMint-address-uint256-}[`_safeMint`], with an
additional `data` parameter which is
     * forwarded in {IERC721Receiver-onERC721Received} to contract
recipients.
     */
    function _safeMint(
        address to,
        uint256 tokenId,
        bytes memory data
    ) internal virtual {
        _mint(to, tokenId);
        require(
            _checkOnERC721Received(address(0), to, tokenId, data),
            "ERC721: transfer to non ERC721Receiver implementer"
        );
    }

    /**
     * @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);

        _afterTokenTransfer(address(0), to, tokenId);
    }

    /**
     * @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);

        _afterTokenTransfer(owner, address(0), tokenId);
    }

    /**
     * @dev Transfers `tokenId` from `from` to `to`.
     *  As opposed to {transferFrom}, this imposes no restrictions
on msg.sender.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     *
```

```
     * Emits a {Transfer} event.
     */
    function _transfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {
        require(ERC721.ownerOf(tokenId) == from, "ERC721: transfer
from incorrect owner");
        require(to != address(0), "ERC721: transfer to the zero
address");

        _beforeTokenTransfer(from, to, tokenId);

        // Clear approvals from the previous owner
        _approve(address(0), tokenId);

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

        emit Transfer(from, to, tokenId);

        _afterTokenTransfer(from, to, tokenId);
    }

    /**
     * @dev Approve `to` to operate on `tokenId`
     *
     * Emits an {Approval} event.
     */
    function _approve(address to, uint256 tokenId) internal
virtual {
        _tokenApprovals[tokenId] = to;
        emit Approval(ERC721.ownerOf(tokenId), to, tokenId);
    }

    /**
     * @dev Approve `operator` to operate on all of `owner` tokens
     *
     * Emits an {ApprovalForAll} event.
     */
```

```
    function _setApprovalForAll(
        address owner,
        address operator,
        bool approved
    ) internal virtual {
        require(owner != operator, "ERC721: approve to caller");
        _operatorApprovals[owner][operator] = approved;
        emit ApprovalForAll(owner, operator, approved);
    }

    /**
     * @dev Reverts if the `tokenId` has not been minted yet.
     */
    function _requireMinted(uint256 tokenId) internal view virtual
{
        require(_exists(tokenId), "ERC721: invalid token ID");
    }

    /**
     * @dev Internal function to invoke
{IERC721Receiver-onERC721Received} on a target address.
     * The call is not executed if the target address is not a
contract.
     *
     * @param from address representing the previous owner of the
given token ID
     * @param to target address that will receive the tokens
     * @param tokenId uint256 ID of the token to be transferred
     * @param data bytes optional data to send along with the call
     * @return bool whether the call correctly returned the
expected magic value
     */
    function _checkOnERC721Received(
        address from,
        address to,
        uint256 tokenId,
        bytes memory data
    ) private returns (bool) {
        if (to.isContract()) {
            try IERC721Receiver(to).onERC721Received(_msgSender(),
from, tokenId, data) returns (bytes4 retval) {
                return retval ==
```

```
IERC721Receiver.onERC721Received.selector;
            } catch (bytes memory reason) {
                if (reason.length == 0) {
                    revert("ERC721: transfer to non ERC721Receiver
implementer");
                } else {
                    /// @solidity memory-safe-assembly
                    assembly {
                        revert(add(32, reason), mload(reason))
                    }
                }
            }
        } else {
            return true;
        }
    }

    /**
     * @dev Hook that is called before any token transfer. This
includes minting
     * and burning.
     *
     * Calling conditions:
     *
     * - When `from` and `to` are both non-zero, ``from``'s
`tokenId` will be
     * transferred to `to`.
     * - When `from` is zero, `tokenId` will be minted for `to`.
     * - When `to` is zero, ``from``'s `tokenId` will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to
xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This
```

```
includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to
xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual {}
}
```

Standard ERC721 Contract

2. IERC721

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0)
(token/ERC721/IERC721.sol)

pragma solidity ^0.8.0;

import "../../utils/introspection/IERC165.sol";

/**
 * @dev Required interface of an ERC721 compliant
contract.
 */
interface IERC721 is IERC165 {
    /**
     * @dev Emitted when `tokenId` token is transferred
from `from` to `to`.
     */
    event Transfer(address indexed from, address indexed
to, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables `approved` to
manage the `tokenId` token.
     */
    event Approval(address indexed owner, address indexed
approved, uint256 indexed tokenId);

    /**
     * @dev Emitted when `owner` enables or disables
(`approved`) `operator` to manage all of its assets.
     */
    event ApprovalForAll(address indexed owner, address
indexed operator, bool approved);

    /**
```

```
     * @dev Returns the number of tokens in ``owner``'s
account.
     */
    function balanceOf(address owner) external view
returns (uint256 balance);

    /**
     * @dev Returns the owner of the `tokenId` token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function ownerOf(uint256 tokenId) external view
returns (address owner);

    /**
     * @dev Safely transfers `tokenId` token from `from`
to `to`.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by
`from`.
     * - If the caller is not `from`, it must be approved
to move this token by either {approve} or
{setApprovalForAll}.
     * - If `to` refers to a smart contract, it must
implement {IERC721Receiver-onERC721Received}, which is
called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
```

```
        address from,
        address to,
        uint256 tokenId,
        bytes calldata data
    ) external;

    /**
     * @dev Safely transfers `tokenId` token from `from`
to `to`, checking first that contract recipients
     * are aware of the ERC721 protocol to prevent tokens
from being forever locked.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must exist and be owned by
`from`.
     * - If the caller is not `from`, it must have been
allowed to move this token by either {approve} or
{setApprovalForAll}.
     * - If `to` refers to a smart contract, it must
implement {IERC721Receiver-onERC721Received}, which is
called upon a safe transfer.
     *
     * Emits a {Transfer} event.
     */
    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) external;

    /**
     * @dev Transfers `tokenId` token from `from` to
`to`.
```

```
     *
     * WARNING: Usage of this method is discouraged, use
{safeTransferFrom} whenever possible.
     *
     * Requirements:
     *
     * - `from` cannot be the zero address.
     * - `to` cannot be the zero address.
     * - `tokenId` token must be owned by `from`.
     * - If the caller is not `from`, it must be approved
to move this token by either {approve} or
{setApprovalForAll}.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) external;

    /**
     * @dev Gives permission to `to` to transfer
`tokenId` token to another account.
     * The approval is cleared when the token is
transferred.
     *
     * Only a single account can be approved at a time,
so approving the zero address clears previous approvals.
     *
     * Requirements:
     *
     * - The caller must own the token or be an approved
operator.
     * - `tokenId` must exist.
     *
```

```
     * Emits an {Approval} event.
     */
    function approve(address to, uint256 tokenId)
external;

    /**
     * @dev Approve or remove `operator` as an operator
for the caller.
     * Operators can call {transferFrom} or
{safeTransferFrom} for any token owned by the caller.
     *
     * Requirements:
     *
     * - The `operator` cannot be the caller.
     *
     * Emits an {ApprovalForAll} event.
     */
    function setApprovalForAll(address operator, bool
_approved) external;
    /**
     * @dev Returns the account approved for `tokenId`
token.
     *
     * Requirements:
     *
     * - `tokenId` must exist.
     */
    function getApproved(uint256 tokenId) external view
returns (address operator);


    function isApprovedForAll(address owner, address
operator) external view returns (bool);
}
```

3. Murasaki7NFT Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.
sol";
import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.so
l";
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract Murasaki7NFT is
    ERC721,
    ERC721Enumerable,
    ERC721Burnable,
    AccessControl
{
    using Strings for uint256;
    string private baseURI;
    bytes32 public constant MINTER_ROLE =
keccak256("MINTER_ROLE");

    constructor(string memory buri) ERC721("Murasaki7NFT",
"M7NFT") {
        baseURI = buri;
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }

    function setBaseURI(string memory newBaseURI)
        public
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(bytes(newBaseURI).length > 0, "invalid base uri");
        baseURI = newBaseURI;
    }

    function safeMint(address to, uint256 tokenId)
        public
```

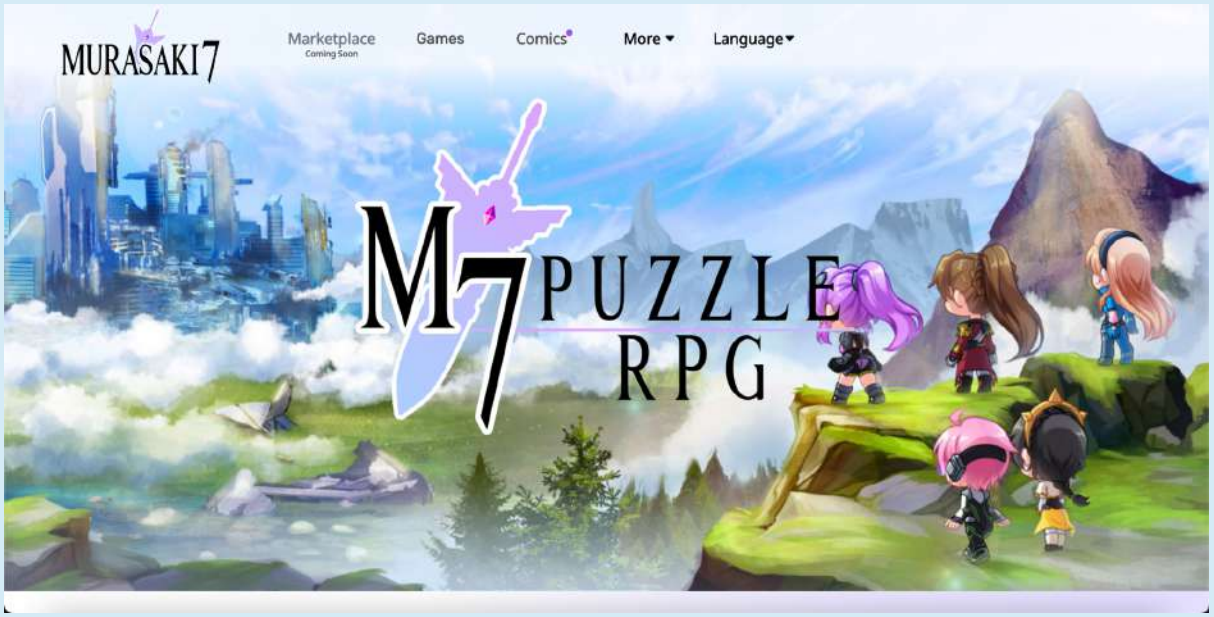
```
        onlyRole(MINTER_ROLE)
    {
        _safeMint(to, tokenId);
    }

    function tokenURI(uint256 tokenId)
        public
        view
        override
        returns (string memory)
    {
        require(_exists(tokenId), "id does not exist");
        return
            string(
                abi.encodePacked(baseURI, tokenId.toString(),
"/metadata.json")
            );
    }

    // The following functions are overrides required by Solidity.

    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal override(ERC721, ERC721Enumerable) {
        super._beforeTokenTransfer(from, to, tokenId);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        override(ERC721, ERC721Enumerable, AccessControl)
        returns (bool)
    {
        return super.supportsInterface(interfaceId);
    }
}
```

WRITE CONTRACT

1. burn
tokenId (uint256)
(Function for burn nft by tokenId value)

2. grantRole
role (byte32)
account (address)
(Function for grant role to address)

WEBSITE REVIEW



- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (By Let's Encrypt SSL)**

Web-Tech stack: React , Cloudflare

Domain .com (cloudflare) - Tracked by whois

First Contentful Paint:	2.3s
Fully Loaded Time	6.5s
Performance	76%
Accessibility	95%
Best Practices	75%
SEO	100%

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees

Note: Please check the disclaimer above and note, that the audit makes no statements or warranties on the business model, investment attractiveness, or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the project owner.