// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

import "@openzeppelin/contracts/utils/Counters.sol";

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract NFT\_Refferal is ERC721 , Ownable {

using Strings for uint256;

using Counters for Counters.Counter;

Counters.Counter internal supply;

uint256 public cost = 0.9 ether;

uint256 public maxSupply = 5000;

string public uriPrefix = "";

uint public \_perBonus = 5;

IERC20 Reward\_Token;

constructor(address \_token) ERC721 ("\_NAME","\_SYMBOL") { //change name and symbol

Reward\_Token = IERC20(\_token);

}

struct User {

address buyer;

string Code;

uint referral;

uint \_bonus\_collected;

}

User[] public userData;

address[] internal buyed;

string[] checker\_ref;

mapping(address => uint) internal total\_purchase;

mapping(string => mapping (address => bool)) internal code\_used;

mapping(string => uint ) internal claimed\_against\_referral;

uint user\_count;

function buy(string memory \_referral) public payable {

if(msg.sender != owner()) {

require(msg.value >= cost,"Insufficient Funds!!");

if(user\_count == 0) {

User memory newUser = User(msg.sender,generateCode(),0,0);

userData.push(newUser);

buyed.push(msg.sender);

total\_purchase[msg.sender] +=1;

user\_count += 1;

}

else if(bytes(\_referral).length > 0 && !BuyedBefore(msg.sender)) {

if(!code\_used[\_referral][msg.sender])

{

code\_used[\_referral][msg.sender] = true;

bool checker = check\_referral(\_referral);

if(!checker) {

require(false,"Code Expired!!");

}

User memory newUser = User(msg.sender,generateCode(),0,0);

userData.push(newUser);

buyed.push(msg.sender);

total\_purchase[msg.sender] +=1;

user\_count += 1;

}

else {

require(false,"Code Already Used!!");

}

}

else if(BuyedBefore(msg.sender) == true && bytes(\_referral).length == 0) {

User memory newUser = User(msg.sender,generateCode(),0,0);

userData.push(newUser);

total\_purchase[msg.sender] +=1;

user\_count += 1;

}

else {

code\_used[\_referral][msg.sender] == true

? require(false,"Referral Already Used!!")

: require(false,"Invalid Parameters!!");

}

}

mint(msg.sender);

}

function BuyedBefore(address \_user) public view returns (bool) {

for (uint i = 0; i < buyed.length; i++) {

if (buyed[i] == \_user) {

return true;

}

}

return false;

}

function check\_referral(string memory referral) internal returns (bool){

for(uint i = 0; i < user\_count ; i++ ) {

if(keccak256(abi.encodePacked((userData[i].Code))) == keccak256(abi.encodePacked((referral))))

{

if(userData[i].referral >= 5) { //total code used

return false;

}

userData[i].referral += 1;

return true;

}

}

require(false, "Invalid Referral!!");

}

function generateCode() internal view returns (string memory) {

return string(abi.encodePacked("STF","-V",supply.current().toString(),user\_count.toString())); //Change Code Accordingly

}

function Claim\_Reward(string memory \_referral) public {

address \_person = msg.sender;

uint total\_refer = 0;

uint index = 0;

uint calculate = 0;

uint totalBonus = 0;

for(uint i = 0; i < user\_count ; i++ ) {

if(keccak256(abi.encodePacked((userData[i].Code))) == keccak256(abi.encodePacked((\_referral))))

{

if(\_person == userData[i].buyer) {

total\_refer = userData[i].referral;

index = i;

break;

}

else {

require(false,"Your are Not Owner of that Referral!!");

}

}

}

require(total\_refer != 0,"Invalid Referral");

calculate = total\_refer - claimed\_against\_referral[\_referral];

require(calculate != 0 , "Already Claimed");

totalBonus = (cost) \* (calculate \* \_perBonus) / 100 ; //percentage

Reward\_Token.transfer(\_person,totalBonus); //transfer

// Reward\_Token.transferFrom(address(this),\_person,totalBonus); //approve

claimed\_against\_referral[\_referral] += calculate;

User storage newdata = userData[index];

newdata.\_bonus\_collected = claimed\_against\_referral[\_referral];

}

function mint(address \_person) private {

supply.increment();

\_safeMint(\_person, supply.current());

}

function tokenURI(uint256 tokenId)

public

view

virtual

override

returns (string memory)

{

require(

\_exists(tokenId),

"ERC721Metadata: URI query for nonexistent token"

);

string memory currentBaseURI = \_baseURI();

return bytes(currentBaseURI).length > 0

? string(abi.encodePacked(currentBaseURI, tokenId.toString(), ".json"))

: "";

}

function set\_Uri(string memory \_uriPrefix) public onlyOwner { //put pinata ipfs://\_\_\_\_CID\_\_\_\_/

uriPrefix = \_uriPrefix;

}

function totalSupply() public view returns (uint) {

return supply.current();

}

function setCost(uint256 \_newCost) public onlyOwner {

cost = \_newCost;

}

function setBonus(uint256 perBonus) public onlyOwner { //Change Percentage from this Function

\_perBonus = perBonus;

}

function withdraw() public onlyOwner {

(bool os, ) = payable(owner()).call{value: address(this).balance}("");

require(os);

Reward\_Token.transfer(owner(),Reward\_Token.balanceOf(address(this)));

}

function \_baseURI() internal view virtual override returns (string memory) {

return uriPrefix;

}

receive() external payable {}

}