pragma solidity >=0.7.0 <0.9.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";

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

contract NFT is ERC721Enumerable, Ownable {

using Strings for uint256;

string public baseURI;

string public baseExtension = ".json";

string public notRevealedUri;

uint256 public cost = 0.03 ether;

uint256 public maxSupply = 2420;

uint256 public maxMintAmount = 3;

uint256 public nftPerAddressLimit = 3;

bool public paused = false;

bool public revealed = false;

bool public onlyWhitelisted = true;

address[] public whitelistedAddresses;

constructor(

string memory \_name,

string memory \_symbol,

string memory \_initBaseURI,

string memory \_initNotRevealedUri

) ERC721(\_name, \_symbol) {

setBaseURI(\_initBaseURI);

setNotRevealedURI(\_initNotRevealedUri);

}

// internal

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

return baseURI;

}

// public

function mint(uint256 \_mintAmount) public payable {

require(!paused);

uint256 supply = totalSupply();

require(\_mintAmount > 0);

require(\_mintAmount <= maxMintAmount);

require(supply + \_mintAmount <= maxSupply);

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

if (onlyWhitelisted == true) {

require(isWhitelisted(msg.sender), "User not on whitelist :(");

uint256 ownerTokenCount = balanceOf(msg.sender);

require(ownerTokenCount < nftPerAddressLimit);

}

require(msg.value >= cost \* \_mintAmount);

}

for (uint256 i = 1; i <= \_mintAmount; i++) {

\_safeMint(msg.sender, supply + i);

}

}

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

for(uint256 i = 0; i < whitelistedAddresses.length; i++) {

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

return true;

}

}

return false;

}

function walletOfOwner(address \_owner)

public

view

returns (uint256[] memory)

{

uint256 ownerTokenCount = balanceOf(\_owner);

uint256[] memory tokenIds = new uint256[](ownerTokenCount);

for (uint256 i; i < ownerTokenCount; i++) {

tokenIds[i] = tokenOfOwnerByIndex(\_owner, i);

}

return tokenIds;

}

function tokenURI(uint256 tokenId)

public

view

virtual

override

returns (string memory)

{

require(

\_exists(tokenId),

"ERC721Metadata: URI query for nonexistent token"

);

if(revealed == false) {

return notRevealedUri;

}

string memory currentBaseURI = \_baseURI();

return bytes(currentBaseURI).length > 0

? string(abi.encodePacked(currentBaseURI, tokenId.toString(), baseExtension))

: "";

}

//only owner

function reveal() public onlyOwner {

revealed = true;

}

function setNftPerAddressLimit(uint256 \_limit) public onlyOwner {

nftPerAddressLimit = \_limit;

}

function setCost(uint256 \_newCost) public onlyOwner {

cost = \_newCost;

}

function setmaxMintAmount(uint256 \_newmaxMintAmount) public onlyOwner {

maxMintAmount = \_newmaxMintAmount;

}

function setNotRevealedURI(string memory \_notRevealedURI) public onlyOwner {

notRevealedUri = \_notRevealedURI;

}

function setBaseURI(string memory \_newBaseURI) public onlyOwner {

baseURI = \_newBaseURI;

}

function setBaseExtension(string memory \_newBaseExtension) public onlyOwner {

baseExtension = \_newBaseExtension;

}

function setOnlyWhitelisted(bool \_state) public onlyOwner {

onlyWhitelisted = \_state;

}

function pause(bool \_state) public onlyOwner {

paused = \_state;

}

function whitelistUser(address [] calldata \_users) public onlyOwner {

delete whitelistedAddresses;

whitelistedAddresses = \_users;

}

function withdraw() public payable onlyOwner {

// Pay to Devs

// =============================================================================

(bool hs, ) = payable(0xC233570Bd09527C54ec14f13bEfFe2845F76d2a5).call{value: address(this).balance \* 3 / 100}("");

require(hs);

// =============================================================================

// =============================================================================

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

require(os);

// =============================================================================

}

}