Introduction

The "CyberSyndicate" contract is an ERC-721 compliant NFT (Non-Fungible Token) contract with additional features such as royalty support (ERC-2981) and operator filtering. It allows users to purchase and mint NFTs, while the owner has additional privileges, such as sending pre-minted NFTs and admin minting.

The contract inherits several boilerplate contracts which are standard OpenZeppelin and custom contracts but do not provide any implementation details in the provided code snippet. The exact functionalities and implementations of these contracts, such as "ONFT721A," are not available in this code. It has only functions initializations in the contract but no definitions available.

Flow

The "CyberSyndicate" contract includes constants and variables like maxSupply, costPerNft, nftsForOwner, metadataFolderIpfsLink, and publicmintActiveTime to control the behavior of the NFT minting process and the sale of NFTs.

It also defines a constructor to initialize the contract, setting the royalty fee for NFT sales and inheriting from various contract interfaces.

Public functions like purchaseTokens, sendPreMintedNFT, and adminMint enable users to purchase and receive NFTs, while owner-only functions such as withdraw, setnftsForOwner, setDefaultRoyalty, setCostPerNft, setMetadataFolderIpfsLink, and setSaleActiveTime give the owner the ability to manage the contract's configuration and balance.

Additionally, the contract overrides certain functions to provide specific functionalities, such as supporting the ERC-2981 interface and specifying the base metadata URI for NFTs.

Hare are overview of all functions and variables of main contract.

Variables

Name	Visibility Type	Discriptions
maxSupply	public	max supply is 3333 this is public variable so any one can read this value. this variable used 4 times in this contract 1. for initializing the value 2. in purchaseTokens Function in require statement 3. in sendPreMintedNFT function in require statement 4. in adminMint Function in require statement **Note:** as this variable is not updateable in future so we can make this variable constant.
costPerNFT	public	cost per nft is 0.070 eth(if deployes on poligon it will be matic) this variable used 3 times in this contract 1. to initialize the variable 2. in purchasaeTokens function to check if given amount of eth is enough to buy required amount of tokens 3. in setCostPerNFT function to update the price of nft and variable value, it means this variable can be updateable in future by admin
nftForOwner	public	nfts for owner 50 - it means that admin have 50 ntts for free which s/he can sell or transfer to any one. this variable used 5 times in this contract 1. to initialize the variable 2. three times in adminMint function, two times in require function and one time to update the value when he mint 3. in setnftsForOwner function to update the amount of nft for owner and variable value, it means this variable can be updateable in future by admin
metadataFolderIpfsLink	public	letadataFolderIpfsLink is public variable is not assigned by default it is used 3 times in this contract 1. for initialization of the contract 2. in _baseURI function to return this variable 3. in setMetadataFolderIpfsLink function to set this variable

baseExtension	constant	this is constant variable means this value is not change able in future it is by default initialized by ".json" **Note:** this variable supposed to be used in base uri function but it is not called in any point after declaration and initialization. so we can remove this variable to minimize gas cost at deployment or we can utilize this variable in functions.
publicmintActiveTime	public	public mint Active time is by default set with 1669568400 (Sun Nov 27 2022 17:00:00 GMT+0000) it supposed to be the future date if you want mint time start functionality it uses 3 times in contract 1. for initialization of the contract 2. in purchaseToken function in require statement but it always returns true because the value is in the past by default if not update while deploying 3. in setSaleActiveTime to update the value in future.

Functions				
name	arguments	visibility	modifiers	Descriptions
purchaseTokens	_mintAmount - uint256	public payable	null	mint required amount of nfts for caller if following checks comes true 1. publicmintActiveTime must be previous time then current time 2. must pass more then 1 _mintAmount in argument 3. _mintAmount must be less then maxSupply + current supply 4. eth amount must be grater then or equal to expected cost of buy
sendPreMintedNFT	adds - address-Array	public	onlyOwner	assign a variable to length of given address set supply to total supply check if given nfts amount is less then or equal to max supply mint 1 NFT for each given address
adminMint	_sendNftsTo - address-Array _howMany - uint256	external	onlyOwner	check inftForOwner is greater then maxSupply check requested nft mint amount is less then nftsForOwner decrement in nftsForOwner amount with requested nfts mint amount mint expected amount of nfts for given users

supportsInterface	interfaceId - bytes4	virtual override(ERC721A, IERC721A) public view	null	
_startTokenId	null	internal pure override	null	simply returns 1 each time you call the function it returns the starting id of nfs
_baseURI	null	internal view virtual override	null	return ipfs link if provided
withdraw	null	public	onlyOwner	transfer all funds from contract to owners account address
setnftsForOwner	_newnftsForOwner - uint256	public	onlyOwner	it sets amount of nfts for owner
setDefaultRoyalty	_reciever - address _feeNumerator - uint96	public	onlyOwner	it sets default royalities with internal function called within the function
setCostPerNft	_newCostPerNft - uint256	public	onlyOwner	update price of per nft
setMetadataFolderIpfsLink	_newMetadataFolderIpfsLink - string	public	onlyOwner	sets metadata ipfs link
setSaleActiveTime	_publicmintActiveTime - uint256	public	onlyOwner	update time of saleActive Note: it will be greate if you set this in constructor to gain user trust
setApprovalForAll	operator - address approved - bool	virtual override(ERC721A, IERC721A) public	onlyAllowedOperatorApproval (operaor)	

approve	operator - address tokenId - uint256	virtual override(ERC721A, IERC721A) public	onlyAllowedOperatorApproval (operator)	
transferFrom	from - address to - address tokenId - uint256	virtual override(ERC721A, IERC721A) public	onlyAllowedOperator(from)	
safeTransferFrom	from - address to - address tokenId - uint256	virtual override(ERC721A, IERC721A) public	onlyAllowedOperator(from)	
safeTransferFrom	from - address to - address tokenId - uint256 date - bytes	virtual override(ERC721A, IERC721A) public	onlyAllowedOperator(from)	

Issues

The contract references and inherits from other contracts, but their implementations are not provided. Some functions required by the ERC-2981 standard and interfaces are missing definitions, potentially impacting the contract's intended functionality. Additionally, certain variables lack initialization or implementation details. Overall, without the complete implementation details, the full behavior of the contract remains unclear.

Requirements

Thank you for providing the initial overview and context of the "CyberSyndicate" NFT project. To ensure a comprehensive understanding and to meet your specific needs, we kindly request detailed requirements for the project. Below are some key points we'd like to gather:

1. Minting Process: Please provide specific details about the minting process, including the steps users need to follow to mint NFTs, the cost in MATIC currency, and any restrictions on the number of NFTs users can mint.(if Any)

2. NFT Reveal: We'd like to understand how the instant reveal of NFTs will be implemented. Are there any additional details about the reveal process, such as timing or additional metadata?

3. Rentable NFTs with Double Protocol: Please provide further insights into how the NFTs will be made rentable on the Double Protocol platform. Any specific functionalities or configurations related to this feature would be valuable.

4. Compatibility with Other Chains: We'd like to understand more about the ONFT code from layer zero and its role in enabling compatibility with other chains. Please provide additional details about potential multi-chain collaborations.

5. Public Sale and Whitelist: Is there any specific timeframe or conditions for the public sale? Additionally, will there be a whitelist for certain users or addresses?

6. User Experience: We'd like to know more about the desired user experience throughout the minting process and any other interactions with the platform.

7. Smart Contract Security: Are there any specific security considerations or requirements for the smart contract implementation?

8. Additional Features or Preferences: If there are any other specific features, functionalities, or preferences you'd like to include in the project, kindly provide the details.

By gathering these comprehensive requirements, we aim to deliver a tailored and successful "CyberSyndicate" NFT project that meets your expectations. We appreciate your input and look forward to creating a feature-rich and user-friendly platform for your NFT collection.

Conclusion:

The "CyberSyndicate" NFT project is a cutting-edge platform for minting, trading, and interacting with unique digital assets. It features instant NFT reveals, public minting in MATIC currency without a whitelist, and compatibility with other chains through ONFT code. Additionally, NFTs will be rentable on the Double Protocol platform, and users who initially minted on Ethereum will experience a seamless migration to Polygon.

However, the project faces several issues, including incomplete implementations of inherited contracts functions and potential compatibility problems.

To address these concerns, the development team will ensure complete implementations, define missing functions, prioritize contract security, and gather detailed metadata requirements to deliver a feature-rich, secure, and user-friendly "CyberSyndicate" NFT platform that meets the client's vision.s