## **Overview**

This Solidity smart contract implements a basic ERC721-compliant Non-Fungible Token (NFT) contract with additional functionalities. The contract allows users to mint new NFTs, set metadata, and configure various parameters. It also includes ownership management using the Ownable contract from the OpenZeppelin library.

## **Contract Details**

* **License**: MIT License
* **Solidity Version Compatibility**: >=0.7.0 <0.9.0
* **Dependencies**: This contract imports two other contracts:
* **ERC721Enumerable.sol**: A standard ERC721 implementation with enumeration support, part of the OpenZeppelin library.
* **Ownable.sol**: A contract from the OpenZeppelin library to manage ownership of the NFT contract.

## **Constructor**

The constructor of the contract is used to initialize the contract and set initial values for important parameters.

### **Parameters**

* **\_name** (string): The name of the NFT collection.
* **\_symbol** (string): The symbol or ticker for the NFT collection.
* **\_initBaseURI** (string): The base URI for metadata of the NFTs.
* **\_initNotRevealedUri** (string): The URI for metadata to display when the NFT has not been revealed yet.

## **State Variables**

* **baseURI** (string): The base URI for NFT metadata, which will be combined with the token ID and **baseExtension** to create the full metadata URI.
* **baseExtension** (string): The file extension for the NFT metadata files (e.g., ".json").
* **cost** (uint256): The cost in ether to mint each NFT.
* **maxSupply** (uint256): The maximum total supply limit for the NFT collection.
* **maxMintAmount** (uint256): The maximum number of NFTs that can be minted in a single transaction.
* **paused** (bool): A flag to indicate whether the minting process is paused (if set to true, minting is not allowed).
* **revealed** (bool): A flag to indicate whether the NFT collection has been revealed.
* **notRevealedUri** (string): The URI for metadata to display when the NFT has not been revealed yet.

## **Modifiers**

* **onlyOwner**: A custom modifier used in various functions to restrict access to the contract owner.

## **Functions**

The contract provides several external functions to interact with and manage the NFT collection:

* **mint(uint256 \_mintAmount)**: Allows users to mint new NFTs. It requires that the minting process is not paused, the minting amount is within the maximum limit, and the total supply doesn't exceed the maximum supply. Non-owner users are also required to send enough ether to cover the minting cost.
* **walletOfOwner(address \_owner)**: Retrieves an array of token IDs owned by a specific address (owner).
* **tokenURI(uint256 tokenId)**: Retrieves the URI for the metadata of a specific token. If the collection has not been revealed, it returns the **notRevealedUri**. Otherwise, it constructs the full metadata URI using the **baseURI**, token ID, and **baseExtension**.
* **reveal()**: Allows the contract owner to reveal the NFT collection by setting the **revealed** flag to true.
* **setCost(uint256 \_newCost)**: Allows the contract owner to set the new minting cost.
* **setmaxMintAmount(uint256 \_newmaxMintAmount)**: Allows the contract owner to set the new maximum minting amount.
* **setNotRevealedURI(string memory \_notRevealedURI)**: Allows the contract owner to set the URI for metadata when the NFT is not revealed yet.
* **setBaseURI(string memory \_newBaseURI)**: Allows the contract owner to set the new base URI for NFT metadata.
* **setBaseExtension(string memory \_newBaseExtension)**: Allows the contract owner to set the new file extension for NFT metadata.
* **pause(bool \_state)**: Allows the contract owner to pause or unpause the minting process.
* **withdraw()**: Allows the contract owner to withdraw the contract's balance. It sends 5% of the balance to address **0x943590A42C27D08e3744202c4Ae5eD55c2dE240D** as support and the remaining 95% to the contract owner.

## **Note**

This documentation provides an overview of the NFT smart contract and its functionalities. Before deploying or interacting with any smart contract, it is crucial to conduct a thorough code review, audit the contract's security, and understand the implications of each function. Additionally, the contract references the OpenZeppelin library, which should be properly imported and verified for use.