# 119.https://stackoverflow.com/questions/72805188/how-to-link-my-ipfs-content-to-erc721-contract

**T:**How to link my IPFS content to ERC721 contract?

**Q:**I´m trying to make a simple ERC721 NFT minting contract. I created an image and its corresponding metadata and I´ve uploaded them to the ipfs. When does this image and metadata link with the token created in the smart contract?. I was trying to use this code generated with the Openzeppelin contract wizard:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.4;import "@openzeppelin/contracts/token/ERC721/ERC721.sol";import "@openzeppelin/contracts/access/Ownable.sol";import "@openzeppelin/contracts/utils/Counters.sol";contract MyToken is ERC721, Ownable { using Counters for Counters.Counter; Counters.Counter private \_tokenIdCounter; constructor() ERC721("MyToken", "MTK") {} function safeMint(address to) public onlyOwner { uint256 tokenId = \_tokenIdCounter.current(); \_tokenIdCounter.increment(); \_safeMint(to, tokenId); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
But I don't see where I can link my nft image and metadata to the contract. I´ve seen that in ERC721 standart by openzeppelin you can set the base URI and tokenURI with the functions tokenURI and \_baseURI, but I don't know exactly how to use them. I was planning to create a multiple-item collection (in the ipfs) so I don't know what to use in my case.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

1 **Answer**

**A1:**The function you wrote is not enough. When you work with NFT, you have to set some state variables. You have to keep track of the listed items, token ids, used token urls. (your contract logic might be different). Also, you have to add some require statements for validation before minting a token. first define your state variables:  
  
 using Counters for Counters.Counter; // initially 0 Counters.Counter private \_listedItems; Counters.Counter private \_tokenIds; uint public listingPrice=0.025 ether; // mapping is similar to object in javascript mapping(string=>bool) private \_usedTokenURIs;  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
then write the minting logic:  
  
function mintToken(string memory tokenURI,uint price) public payable returns (uint){ // make sure you do not mint same uri require( \_usedTokenURIs[tokenURI]==false,"Token URI already exists"); // since this function is payable, the amount that you sent is stored in msg.value by ethereum evm require(msg.value==listingPrice,"Price must be equal to listing fee"); \_tokenIds.increment(); \_listedItems.increment(); uint newTokenId=\_tokenIds.current(); // this is a wrapper for \_mint \_safeMint(msg.sender,newTokenId); \_setTokenURI(newTokenId, tokenURI); \_createNftItem(newTokenId,price); // update \_usedTokenURIs mapping \_usedTokenURIs[tokenURI]=true; return newTokenId; }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Now to store the ipfs link in the blockchain you have to call mintToken function on the front end. You create a form, add inputs related to the metadata, and post the json metadata and image to the ipfs. When you post the data you get the nft uri. Store the nft uri in the state. Then you write a createNft function on the front end. Once you successfully call the contract.mintToken function, your ipfs link will be stored on blockchain.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]