# 287.https://stackoverflow.com/questions/71783185/how-can-i-implement-a-lazy-minting-smart-contract-that-doesnt-actually-mint-nft

**T:**How can I implement a lazy-minting smart contract that doesn't actually mint NFTs?

**Q:**This is my first experience writing a smart contract, so I am still trying to navigate how this all works. I have a smart contract on the Rinkeby test network that currently works as expected when deploying and then importing into OpenSea (their "get listed" page).  
  
The problem is I have to mint NFTs within the constructor of the smart contract by calling the \_mint() method to make them display in the collection that OpenSea generates based on my smart contract's address. I have only deployed the test smart contract with minting the first 10 NFTs of the collection and those 10 NFTs show up on OpenSea as I would expect. However, I want to utilize OpenSea's lazy minting ability to avoid the gas fees for minting each NFT(there are 10,000 of them) in the collection myself.  
  
Is there a way to set up the smart contract that informs OpenSea(and possibly other exchanges) that I am lazy-minting and have all 10,000 NFTs display within OpenSea without actually having to mint each one upfront within the contract?  
  
I know people can develop their own site where others can then mint each NFT they would like to purchase to accomplish lazy-minting, but I want these NFTs to all be visible and purchasable within OpenSea without them officially being minted yet(lazy-minted). I have all my NFTs and metadata out on IPFS and everything seems to be linking up properly when I perform an actual mint action, so I know that works just fine. I just need to understand if/how it is possible to set up my smart contract to enable this functionality.  
  
Here is my current smart contract with the URLs to IPFS replaced with a placeholder:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.4;import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";import "@openzeppelin/contracts/access/Ownable.sol";import "@openzeppelin/contracts/security/Pausable.sol";import "@openzeppelin/contracts/utils/math/SafeMath.sol";import "@openzeppelin/contracts/interfaces/IERC2981.sol";import "./ContextMixin.sol";contract MyTestContract is ERC1155, IERC2981, Ownable, Pausable, ContextMixin { using SafeMath for uint256; string public name; string public symbol; uint256 public total\_supply; address private \_recipient; constructor() ERC1155("IPFS\_URL\_TO\_JSON\_FILES\_GOES\_HERE") { name = "MY Collection Name Goes Here"; symbol = "TESTING"; total\_supply = 10000; \_recipient = owner(); \_mint(msg.sender, 1, 1, ""); \_mint(msg.sender, 2, 1, ""); \_mint(msg.sender, 3, 1, ""); \_mint(msg.sender, 4, 1, ""); \_mint(msg.sender, 5, 1, ""); \_mint(msg.sender, 6, 1, ""); \_mint(msg.sender, 7, 1, ""); \_mint(msg.sender, 8, 1, ""); \_mint(msg.sender, 9, 1, ""); \_mint(msg.sender, 10, 1, ""); } function setURI(string memory newuri) public onlyOwner { \_setURI(newuri); } function pause() public onlyOwner { \_pause(); } function unpause() public onlyOwner { \_unpause(); } function mint(address account, uint256 id, uint256 amount, bytes memory data) public onlyOwner { \_mint(account, id, amount, data); } function mintBatch(address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data) public onlyOwner { \_mintBatch(to, ids, amounts, data); } function \_beforeTokenTransfer(address operator, address from, address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data) internal whenNotPaused override { super.\_beforeTokenTransfer(operator, from, to, ids, amounts, data); } function royaltyInfo(uint256 \_tokenId, uint256 \_salePrice) external view override returns (address receiver, uint256 royaltyAmount) { return (\_recipient, (\_salePrice \* 500) / 10000); } function \_setRoyalties(address newRecipient) internal { require(newRecipient != address(0), "Royalties: new recipient is the zero address"); \_recipient = newRecipient; } function setRoyalties(address newRecipient) external onlyOwner { \_setRoyalties(newRecipient); } function supportsInterface(bytes4 interfaceId) public view virtual override(ERC1155, IERC165) returns (bool) { return (interfaceId == type(IERC2981).interfaceId || super.supportsInterface(interfaceId)); } /\*\* \* Override isApprovedForAll to auto-approve OS's proxy contract \*/ function isApprovedForAll( address \_owner, address \_operator ) public override view returns (bool isOperator) { // if OpenSea's ERC1155 Proxy Address is detected, auto-return true if (\_operator == address(0x207Fa8Df3a17D96Ca7EA4f2893fcdCb78a304101)) { return true; } // otherwise, use the default ERC1155.isApprovedForAll() return ERC1155.isApprovedForAll(\_owner, \_operator); } /\*\* \* This is used instead of msg.sender as transactions won't be sent by the original token owner, but by OpenSea. \*/ function \_msgSender() internal override view returns (address sender) { return ContextMixin.msgSender(); } // Update for collection-specific metadata. function contractURI() public pure returns (string memory) { return "CONTRACT\_LEVEL\_METADATA\_URL\_GOES\_HERE"; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Any help would be greatly appreciated.

2 **Answer**

**A1:**No, that is not possible for now. When there is a website you go to mint NFTs for yourself, the website is doing the following:  
  
 ● You click mint and then the NFT is minted on the contract to your account.  
When you go to Opensea, they are able to fetch the NFTs on your account and then to load their data.  
  
There is, for right now, no implementation of them getting the data without the data being associated with a EOA or a Contract Account.  
  
Basically, the NFT must be someone's. When you have the data, but no a owner, it's not someone's.

**A2:**I think that what you want to do is to display a "lazy minted" NFT in one of the marketplaces right? If that's what you need, maybe this article will help you? You upload your lazy minted data to Rarible using rarible's API, and then the NFT can be viewed and minted on their site. I'm not sure if Openseas provide an API to do this too. Once the NFT is minted, I guess it can be seen in either Openseas or Rarible.