# 230.https://stackoverflow.com/questions/72190423/erc721-contract-deployed-with-create2-stops-working-after-a-particular-function

**T:**ERC721 contract deployed with create2 stops working after a particular function is called

**Q:**So I'm working on an NFT marketplace smart contract and one of the features in this marketplace allows users to deploy their own collection (this is basically a smart contract that extends the ERC721 specification). For this, I'm leveraging the create2 opcode. It deploys well while testing on the testnet (except for the insanely huge gas requirement) but the problem is, I can only call a function once in this contract. This is the contract that extends ERC721:  
  
pragma solidity ^0.8.0;import '@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol';import '@openzeppelin/contracts/token/ERC721/IERC721.sol';import '@openzeppelin/contracts/security/ReentrancyGuard.sol';import '@openzeppelin/contracts/utils/Counters.sol';import './interfaces/IDeployableCollection.sol';contract DeployableCollection is IDeployableCollection, ERC721URIStorage, ReentrancyGuard { using Counters for Counters.Counter; Counters.Counter private \_tokenIds; address public \_collectionOwner; bytes32 public \_category; address payable public \_paymentReceiver; string public \_collectionURI; mapping(address => uint256) public lastMintedForIDs; constructor( string memory name\_, string memory symbol\_, address collectionOwner\_, string memory category\_, address paymentReceiver\_, string memory collectionURI\_ ) ERC721(name\_, symbol\_) { \_collectionOwner = collectionOwner\_; \_category = keccak256(abi.encode(category\_)); \_paymentReceiver = payable(paymentReceiver\_); \_collectionURI = collectionURI\_; } function mintFor(string memory \_tokenURI, address to) external nonReentrant returns (uint256 \_tokenId) { \_tokenIds.increment(); \_tokenId = \_tokenIds.current(); \_mint(to, \_tokenId); \_setTokenURI(\_tokenId, \_tokenURI); lastMintedForIDs[to] = \_tokenId; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This is the line that deploys the contract:  
  
function deployCollection( string memory name\_, string memory symbol\_, string memory category\_, address paymentReceiver\_, string memory \_collectionURI ) external payable nonReentrant { uint256 \_fee = \_utilityToken != address(0) && IERC20(\_utilityToken).balanceOf(\_msgSender()) >= \_requiredHold ? \_collectionDeployFeeInEther.sub((uint256(\_percentageDiscount).mul(\_collectionDeployFeeInEther)).div(100)) : \_collectionDeployFeeInEther; require(msg.value >= \_fee, 'FEE\_TOO\_LOW'); bytes memory \_byteCode = abi.encodePacked( type(DeployableCollection).creationCode, abi.encode(name\_, symbol\_, \_msgSender(), category\_, paymentReceiver\_, \_collectionURI) ); bytes32 \_salt = keccak256(abi.encode(name\_, \_msgSender())); address \_collection; assembly { \_collection := create2(0, add(\_byteCode, 32), mload(\_byteCode), \_salt) } emit CollectionDeployed(\_collection, \_msgSender(), block.timestamp, name\_, category\_, symbol\_); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Now this works fine, except once I call mintFor and an NFT gets minted, the contract stops working. It shows a total supply of zero (0) on the block explorer even though an NFT has been created (I'm only able to create an NFT with an ID of 1). I also am not able to call any function again as it raises an exception (the cause of which is unknown). This is the line in the factory contract that does the actual minting:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 function mintNFT( address collection, string memory tokenURI\_, address \_for ) external payable nonReentrant returns (bool) { uint256 \_fee = \_utilityToken != address(0) && IERC20(\_utilityToken).balanceOf(\_msgSender()) >= \_requiredHold ? \_mintFeeInEther.sub((uint256(\_percentageDiscount).mul(\_mintFeeInEther)).div(100)) : \_mintFeeInEther; require(msg.value >= \_fee, 'FEE\_TOO\_LOW'); address \_paymentReceiver = IDeployableCollection(collection).\_paymentReceiver(); uint256 \_feeForOwner = (uint256(\_percentageForCollectionOwners).mul(\_fee)).div(100); \_safeMintFor(collection, tokenURI\_, \_for); \_safeTransferETH(\_paymentReceiver, \_feeForOwner); uint256 \_tokenId = IDeployableCollection(collection).lastMintedForIDs(\_msgSender()); emit Mint(collection, \_tokenId, block.timestamp, tokenURI\_, \_msgSender()); return true; }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
I'm guessing an optimization with a runs of 1 is applied upon deployment using create2 (forgive me if this is stupid) but I also think it isn't likely as I'm also watching for events on the smart contract with a Nodejs back-end and I'm able to call '\_collectionURI()' when the event data is propagated to my back-end app and this is before calling mintFor. I'm confused! Please help!  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
This is a link to the contract info on the explorer: https://testnet.bscscan.com/token/0x6dd5bd0072cdc5e8c24f262a9631c175bc2356a0

0 **Answer**