# 5.https://stackoverflow.com/questions/71539232/setapprovalforall-not-owner-nor-approved

**T:**SetApprovalForAll: not owner nor approved

**Q:**Hello to everyone i have a problem in my contracts and i don't understand how to solve it...Is a Nft Marketplace, and the problem happen on the second sale, i mean if i mint a token, then import the token in the markeplace and sell it from A to B is working....but if i try to sell from B to C i receive error "Caller not owner nor approved".  
  
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/extensions/ERC721URIStorage.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Counters.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol";import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";pragma solidity ^0.8.0;contract NFTS is ERC721URIStorage, Ownable{ using Counters for Counters.Counter; Counters.Counter private \_tokenId; address contractOwner; address marketPlace; constructor(address \_marketplace)ERC721("AAA", "BBB"){ marketPlace = \_marketplace; contractOwner = msg.sender; } mapping(uint=>address) public tokenCreator; function createNFT(string memory \_uri)external returns(uint){ \_tokenId.increment(); uint newId = \_tokenId.current(); \_safeMint(msg.sender, newId); \_setTokenURI(newId, \_uri); setApprovalForAll(marketPlace, true); tokenCreator[newId] = msg.sender; return newId; }}contract Marketplace is Ownable, ReentrancyGuard{ using Counters for Counters.Counter; Counters.Counter private \_itemId; address payable contractOwner; uint listingPrice = 1 wei; uint delistingPrice = 1 wei; constructor(){ contractOwner = payable(msg.sender); } struct importItemInMarketplace{ address \_nftContract; address payable contractOwner; address payable tokenCreator; address payable tokenOwner; address payable buyer; uint tokenId; uint price; uint royalties; } mapping(uint=>importItemInMarketplace) public idItemImportedInMarketplace; event successImported( address \_nftContract, address payable contractOwner, address payable tokenCreator, address payable tokenOwner, address payable buyer, uint tokenId, uint price, uint royalties ); struct itemForRoyalties{ address payable contractOwner; address payable tokenCreator; uint price; uint royalties; } mapping(uint=>itemForRoyalties) public tokenIdRoyalties; function itemInMarketplace( address \_nftContract, uint \_tokenId, uint \_price, uint \_royalties )external payable nonReentrant{ address tokenOwner = NFTS(\_nftContract).ownerOf(\_tokenId); address tokenCreator = NFTS(\_nftContract).tokenCreator(\_tokenId); require(\_price > 0, "Cannot set a price less or equal to 0!"); require(msg.value == listingPrice, "You have to pay 1 wei to list your token!"); require(msg.sender == tokenOwner, "Cannot list a token if is not yours!" ); require(\_royalties > 5 && \_royalties < 20, "You can set royalties between 5% and 20 % !"); \_itemId.increment(); uint newId = \_itemId.current(); idItemImportedInMarketplace[newId] = importItemInMarketplace( \_nftContract, payable(contractOwner), payable(tokenCreator), payable(tokenOwner), payable(address(0)), \_tokenId, \_price, \_royalties ); tokenIdRoyalties[\_tokenId] = itemForRoyalties( payable(contractOwner), payable(tokenCreator), \_price, \_royalties ); } function getDataRoyalties(uint \_tokenId)public view returns( address, address, uint, uint ){ return (tokenIdRoyalties[\_tokenId].contractOwner, tokenIdRoyalties[\_tokenId].tokenCreator, tokenIdRoyalties[\_tokenId].price, tokenIdRoyalties[\_tokenId].royalties); } struct ItemSold{ address \_nftContract; address payable contractOwner; address payable tokenCreator; address payable prevTokenOwner; address payable buyer; uint tokenId; uint price; uint royalties; } mapping(uint=>ItemSold)public soldedItemTokenId; event SoldSuccess( address \_nftContract, address payable contractOwner, address payable tokenCreator, address payable prevTokenOwner, address payable buyer, uint tokenId, uint price, uint royalties ); function purchase(address \_nftContract, uint \_itemIds)external payable nonReentrant{ address payable \_contractOwner = idItemImportedInMarketplace[\_itemIds].contractOwner; address payable \_tokenOwner = idItemImportedInMarketplace[\_itemIds].tokenOwner; address payable \_tokenCreator = idItemImportedInMarketplace[\_itemIds].tokenCreator; address payable \_prevTokenOwner = \_tokenOwner; address payable \_buyer = idItemImportedInMarketplace[\_itemIds].buyer; uint \_tokenId = idItemImportedInMarketplace[\_itemIds].tokenId; uint \_price = idItemImportedInMarketplace[\_itemIds].price; uint \_royalties = idItemImportedInMarketplace[\_itemIds].royalties; require(msg.sender != \_contractOwner && msg.sender != \_tokenOwner, "Admin or tokenOwner cannot buy this Item!"); require(msg.value == \_price, "Set the right price to buy it!"); payable(\_contractOwner).transfer(listingPrice); \_tokenOwner = payable(msg.sender); payable(\_prevTokenOwner).transfer(\_price); \_buyer = payable(\_tokenOwner); NFTS(\_nftContract).transferFrom(\_prevTokenOwner, \_buyer, \_tokenId); delete(idItemImportedInMarketplace[\_itemIds]); \_itemId.decrement(); soldedItemTokenId[\_tokenId] = ItemSold( \_nftContract, payable(\_contractOwner), payable(\_tokenCreator), payable(\_prevTokenOwner), payable(\_buyer), \_tokenId, \_price, \_royalties ); emit SoldSuccess( \_nftContract, payable(\_contractOwner), payable(\_tokenCreator), payable(\_prevTokenOwner), payable(\_buyer), \_tokenId, \_price, \_royalties ); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Now i don't understand why i receive this error because on the second sale, the B account, can list the token, and by listing function only the token owner can list the token.Any help is appreciate...thanks a lot and have a nice day

**C1:**how did you end up fixing it?

**C2:**Hello @hammies i didn't see your question.

1 **Answer**

**A1:**When a user calls SetApprovalForAll(marketplace, true) in the NFTS contract, he gives permission to the marketplace to trade any of his NFTs minted using that contract.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Now when A transfers the NFT to B, the marketplace executes the order on A's behalf and since A called SetApprovalForAll while minting the token, the marketplace has the permission to transfer A's tokens and there are no issues.  
  
Now when B tries to transfer the NFT to C, B hasn't yet given permission to the marketplace to transfer his tokens. So this part of the code inside your purchase function throws the error "Caller not owner nor approved"  
  
NFTS(\_nftContract).transferFrom(\_prevTokenOwner, \_buyer, \_tokenId);  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
So to solve this, you have to find a way for B to call SetApprovalForAll from the frontend. You can do this when B is signing up to your website or before the token transfer occurs.  
  
With ethers.js, it looks like this  
  
const contract = new ethers.Contract(nftAddress, NFT.abi, signer) //signer is Bconst isApprovedForAll = await contract.isApprovedForAll(signerAddress, marketplaceAddress)if(!isApprovedForAll){ await contract.setApprovalForAll(maketplaceAddress, true)}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]