# 4.https://stackoverflow.com/questions/69779171/this-function-reverted-with-reason-string-erc721-transfer-caller-is-not-owner

**T:**This function reverted with reason string 'ERC721: transfer caller is not owner nor approved'

**Q:**Code:  
  
function f(address nftContract, uint256 itemId, uint256 price) public payable nonReentrant { uint tokenId = idToMarketItem[itemId].tokenId; IERC721(nftContract).approve(address(this), tokenId); IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId);}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
I really dont know whats wrong. Please help me

**C1:**See this answer : ethereum.stackexchange.com/a/112538/84305 I posted it about ERC-20 but the allowance mechanism is similar for ERC-721, the main problem is the assumption that msg.sender is preserved from function f to IERC721(nftContract).approve but it is not.

**C2:**@hroussille Sorry, I didn't quite understand what I need to do. Could you show me please how I need to fix this function?

1 **Answer**

**A1:**The fix is not to do the approve in your function:  
  
function f(address nftContract, uint256 itemId, uint256 price) public payable nonReentrant { uint tokenId = idToMarketItem[itemId].tokenId; IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId);}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Because you are trying to have your contract approve itself to spend funds belonging to the caller (see the answer I linked in the comment, that explains why msg.sender is not what you think inside the IERC721(nftContract).approve call) this will not work. The user must directly call approve on the nftContract.  
  
This is the user's nft, only it should have the right to approve their spending by a third party, no one else.  
  
EDIT : Adding a more complete example / explanation.  
  
Let's take a simple ERC721 contract :  
  
// SPDX-License-Identifier: GPL-3.0pragma solidity >=0.8.0 <0.9.0;import "@openzeppelin/contracts/token/ERC721/ERC721.sol";contract myToken is ERC721 { constructor() ERC721("TOKEN NAME", "TOKEN SYMBOL") { } function mint(uint256 tokenId) public { \_mint(msg.sender, tokenId); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
It exposes all the ERC-721 function with our special function mint(uint256 tokenId).  
  
Now to get closer to your use case let's take a sample smart contract that will try to transfer NFT belonging to a user by means of approval and transferFrom.  
  
// SPDX-License-Identifier: GPL-3.0pragma solidity >=0.8.0 <0.9.0;import "./token.sol";contract myContract { myToken \_tokenContract; constructor(myToken tokenContract) { \_tokenContract = tokenContract; } function requestTransfer(address from, address to, uint256 tokenId) public{ \_tokenContract.transferFrom(from, to, tokenId); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
We send the address of our NFT contract to the constructor at deployement. The only function that we will be interested in is the requestTransfer one.  
  
And now let me add a truffle test file (far from perfect) :  
  
const truffleAssert = require('truffle-assertions');const tokenContract = artifacts.require("myToken");const smartContract = artifacts.require("myContract");contract("myContract", accounts => { it("shound hook the deployed token contract", async () => { const tokenInstance = await tokenContract.deployed(); }); it("shound hook the deployed smart contract", async () => { const contractInstance = await smartContract.deployed(); }); it("should mint 1 NFT", async () => { const tokenInstance = await tokenContract.deployed(); const txResult = await tokenInstance.mint(0, {from: accounts[0]}); truffleAssert.eventEmitted(txResult, "Transfer", (event) => { return event.tokenId == 0; }) }); it("Should fail because smart contract was not approved", async () => { const contractInstance = await smartContract.deployed(); const tokenId = 0; let failed = false; try { const txResult = await contractInstance.requestTransfer(accounts[0], accounts[1], tokenId); } catch (error) { failed = true; } assert.equal(failed, true, "This test should have failed"); }) it("Approves the smart contract to transfer accounts[0] token : (tokenId = 0)", async () => { const contractInstance = await smartContract.deployed(); const tokenInstance = await tokenContract.deployed(); const tokenId = 0; const approved = contractInstance.address; const txResult = await tokenInstance.approve(contractInstance.address, tokenId, {from: accounts[0]}); truffleAssert.eventEmitted(txResult, "Approval", (event) => { return event.tokenId == 0; }) }); it("Should work because smart contract was approved", async () => { const contractInstance = await smartContract.deployed(); const tokenId = 0; let succeded = false; try { const txResult = await contractInstance.requestTransfer(accounts[0], accounts[1], tokenId); succeded = true; } catch (error) { console.log(error); } assert.equal(succeded, true, "This test should have succeded"); })});  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
If you follow the tests :  
  
 ● checks deployement of the token contract  
  
  
 ● checks deployement of the token contract  
  
  
 ● mint 1 NFT with tokenId == 0 for accounts[0]  
  
  
 ● calls the requestTransfer method of our smart contract and fails AS EXPECTED. Because no approval was made for our smart contract to be able to manage tokenId 0 which belongs to accounts[0]  
  
  
 ● Do the approval from accounts[0] address DIRECTLY to the token contract. Allowing our smart contract address to manage tokenId 0.  
  
  
 ● calls the requestTransfer method as in 4) and this time it is expected to succeed because proper approval was made.  
  
  
Only the owner approves a third party to manage its asset(s).I hope it's clear now why test 4) fails. Why 5) is required in order to have 6) succeed. Plus now you have an example for calling approve from JavaScript which would be somewhat similar to what should happen on the client (NFT owner) side.

**C1:**If I do so, the error persists. Sorry for my stupidity, but I don't understand where I need to do approve. Sorry to waste your time on such things, I am very grateful to you.

**C2:**From your user account, did you call the nftContract approve function by giving the address of your smart contract ? The smart contract will be refused spending anything on behalf of the user if it is not approved by the user.

**C3:**And how do I make it so that it is approved by the user? Sorry for such stupid questions

**C4:**It's okay it can be confusing sometimes. check this : ethereum.stackexchange.com/questions/112644/… it's a similar question maybe it's clearer for your. To make it so that it is approved by the user, the user must send a transaction directly to the ERC-20 contract calling on the approve function. This would be done in JavaScript using web3.js on your frontend.

**C5:**Do you mean that I need to use web3.approve(address(this), tokenId) before calling this function in js?