# 116.https://stackoverflow.com/questions/72823707/anchor-test-transaction-failed-with-an-account-required-by-the-instruction-is-m

**T:**Anchor test transaction failed with "An account required by the instruction is missing"

**Q:**I am trying to code an anchor code where I can mint a new edition nft from a master edition ntf. I looked into the metaplex documentation and rust api documentation to come up with the code below, and am not sure why I am getting this "Error: failed to send transaction: Transaction simulation failed: Error processing Instruction 0: An account required by the instruction is missing" message.  
  
For your reference, here I provide both my program script in lib.rs and test script in ts file  
  
lib.rs  
  
use anchor\_lang::prelude::\*;use anchor\_lang::solana\_program::program::invoke;use anchor\_spl::token;use anchor\_spl::token::{MintTo, Token};use mpl\_token\_metadata::instruction::{create\_master\_edition\_v3, create\_metadata\_accounts\_v2,mint\_new\_edition\_from\_master\_edition\_via\_token,};declare\_id!("4kYKrkgyuRppwg6bxxKkirQ5wtddZQpkLA2Xp9Fo2YNn");#[program]pub mod solana\_blockchain { use super::\*; pub fn mint\_nft(ctx: Context<MintNFT>) -> Result<()> { msg!("Initializing Mint NFT"); let cpi\_accounts = MintTo { mint: ctx.accounts.mint.to\_account\_info(), to: ctx.accounts.token\_account.to\_account\_info(), authority: ctx.accounts.payer.to\_account\_info(), }; msg!("CPI Accounts Assigned"); let cpi\_program = ctx.accounts.token\_program.to\_account\_info(); msg!("CPI Program Assigned"); let cpi\_ctx = CpiContext::new(cpi\_program, cpi\_accounts); msg!("CPI Context Assigned"); token::mint\_to(cpi\_ctx, 1)?; msg!("Token Minted !!!"); Ok(()) } pub fn create\_metadata\_account( ctx: Context<CreateMetadataAccount>, creator\_key: Pubkey, uri: String, title: String, symbol: String, ) -> Result<()> { msg!("Initializing the creation of a new metadata account"); let account\_info = vec![ ctx.accounts.metadata.to\_account\_info(), ctx.accounts.mint.to\_account\_info(), ctx.accounts.mint\_authority.to\_account\_info(), ctx.accounts.payer.to\_account\_info(), ctx.accounts.token\_metadata\_program.to\_account\_info(), ctx.accounts.token\_program.to\_account\_info(), ctx.accounts.system\_program.to\_account\_info(), ctx.accounts.rent.to\_account\_info(), ]; msg!("Account Info Assigned"); let creator = vec![ mpl\_token\_metadata::state::Creator { address: creator\_key, verified: false, share: 100, }, mpl\_token\_metadata::state::Creator { address: ctx.accounts.mint\_authority.key(), verified: false, share: 0, }, ]; msg!("Creator Assigned"); // let symbol = std::string::ToString::to\_string("LWB"); invoke( &create\_metadata\_accounts\_v2( ctx.accounts.token\_metadata\_program.key(), ctx.accounts.metadata.key(), ctx.accounts.mint.key(), ctx.accounts.mint\_authority.key(), ctx.accounts.payer.key(), ctx.accounts.payer.key(), title, symbol, uri, Some(creator), 1, true, false, None, None, ), account\_info.as\_slice(), )?; msg!("Metadata Account Created !!!"); // let master\_edition\_infos = vec![ // ctx.accounts.master\_edition.to\_account\_info(), // ctx.accounts.mint.to\_account\_info(), // ctx.accounts.mint\_authority.to\_account\_info(), // ctx.accounts.payer.to\_account\_info(), // ctx.accounts.metadata.to\_account\_info(), // ctx.accounts.token\_metadata\_program.to\_account\_info(), // ctx.accounts.token\_program.to\_account\_info(), // ctx.accounts.system\_program.to\_account\_info(), // ctx.accounts.rent.to\_account\_info(), // ]; // msg!("Master Edition Account Infos Assigned"); // invoke( // &create\_master\_edition\_v3( // ctx.accounts.token\_metadata\_program.key(), // ctx.accounts.master\_edition.key(), // ctx.accounts.mint.key(), // ctx.accounts.payer.key(), // ctx.accounts.mint\_authority.key(), // ctx.accounts.metadata.key(), // ctx.accounts.payer.key(), // None, // ), // master\_edition\_infos.as\_slice(), // )?; // msg!("Master Edition Nft Minted !!!"); Ok(()) } pub fn create\_master\_edition( ctx: Context<CreateMasterEdition>, max\_supply: Option<u64> ) -> Result<()> { let master\_edition\_infos = vec![ ctx.accounts.master\_edition.to\_account\_info(), ctx.accounts.mint.to\_account\_info(), ctx.accounts.mint\_authority.to\_account\_info(), ctx.accounts.payer.to\_account\_info(), ctx.accounts.metadata.to\_account\_info(), ctx.accounts.token\_metadata\_program.to\_account\_info(), ctx.accounts.token\_program.to\_account\_info(), ctx.accounts.system\_program.to\_account\_info(), ctx.accounts.rent.to\_account\_info(), ]; msg!("Master Edition Account Infos Assigned"); invoke( &create\_master\_edition\_v3( ctx.accounts.token\_metadata\_program.key(), ctx.accounts.master\_edition.key(), ctx.accounts.mint.key(), ctx.accounts.payer.key(), ctx.accounts.mint\_authority.key(), ctx.accounts.metadata.key(), ctx.accounts.payer.key(), max\_supply, ), master\_edition\_infos.as\_slice(), )?; msg!("Master Edition Nft Minted !!!"); Ok(()) } pub fn create\_new\_edition\_nft( ctx: Context<CreateNewEdition>, en: u64, ) -> Result<()> { let edition\_infos = vec![ ctx.accounts.token\_program.to\_account\_info(), ctx.accounts.new\_metadata.to\_account\_info(), ctx.accounts.new\_edition.to\_account\_info(), ctx.accounts.master\_edition.to\_account\_info(), ctx.accounts.new\_mint.to\_account\_info(), ctx.accounts.new\_mint\_authority.to\_account\_info(), ctx.accounts.payer.to\_account\_info(), ctx.accounts.token\_account\_owner.to\_account\_info(), ctx.accounts.token\_account.to\_account\_info(), ctx.accounts.new\_metadata\_update\_authority.to\_account\_info(), ctx.accounts.metadata.to\_account\_info(), ctx.accounts.system\_program.to\_account\_info(), ctx.accounts.rent.to\_account\_info(), ]; msg!("Edition Account Infos Assigned"); invoke(&mint\_new\_edition\_from\_master\_edition\_via\_token( ctx.accounts.token\_program.key(),ctx.accounts.new\_metadata.key(),ctx.accounts.new\_edition.key(), ctx.accounts.master\_edition.key(), ctx.accounts.new\_mint.key(),ctx.accounts.new\_mint\_authority.key(), ctx.accounts.payer.key(), ctx.accounts.token\_account\_owner.key(), ctx.accounts.token\_account.key(), ctx.accounts.new\_metadata\_update\_authority.key(), ctx.accounts.metadata.key(), ctx.accounts.original\_mint.key(), en ), edition\_infos.as\_slice())?; msg!("A New Edition Nft Minted !!!"); Ok(()) }}#[derive(Accounts)]pub struct MintNFT<'info> { /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub mint: UncheckedAccount<'info>, // #[account(mut)] pub token\_program: Program<'info, Token>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub token\_account: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub payer: AccountInfo<'info>,}#[derive(Accounts)]pub struct CreateMetadataAccount<'info> { #[account(mut)] pub mint\_authority: Signer<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub mint: UncheckedAccount<'info>, // #[account(mut)] pub token\_program: Program<'info, Token>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub metadata: UncheckedAccount<'info>, // /// CHECK: This is not dangerous because we don't read or write from this account // #[account(mut)] // pub token\_account: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account pub token\_metadata\_program: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub payer: AccountInfo<'info>, pub system\_program: Program<'info, System>, /// CHECK: This is not dangerous because we don't read or write from this account pub rent: AccountInfo<'info>, // /// CHECK: This is not dangerous because we don't read or write from this account // #[account(mut)] // pub master\_edition: UncheckedAccount<'info>,}#[derive(Accounts)]pub struct CreateMasterEdition<'info> { #[account(mut)] pub mint\_authority: Signer<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub mint: UncheckedAccount<'info>, // #[account(mut)] pub token\_program: Program<'info, Token>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub metadata: UncheckedAccount<'info>, // /// CHECK: This is not dangerous because we don't read or write from this account // #[account(mut)] // pub token\_account: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account pub token\_metadata\_program: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub payer: AccountInfo<'info>, pub system\_program: Program<'info, System>, /// CHECK: This is not dangerous because we don't read or write from this account pub rent: AccountInfo<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub master\_edition: UncheckedAccount<'info>,}#[derive(Accounts)]pub struct CreateNewEdition<'info> { /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub original\_mint: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_metadata: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_edition: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub master\_edition: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_mint: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub edition\_mark\_pda: UncheckedAccount<'info>, #[account(mut)] pub new\_mint\_authority: Signer<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub payer: AccountInfo<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub token\_account\_owner: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub token\_account: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_metadata\_update\_authority: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub metadata: UncheckedAccount<'info>, pub token\_program: Program<'info, Token>, pub system\_program: Program<'info, System>, /// CHECK: This is not dangerous because we don't read or write from this account pub rent: AccountInfo<'info>,}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
test file script  
  
import \* as anchor from "@project-serum/anchor";import { Program, Wallet } from "@project-serum/anchor";const assert = require("assert");import { createAssociatedTokenAccountInstruction, createInitializeMintInstruction, getAssociatedTokenAddress, MINT\_SIZE, TOKEN\_PROGRAM\_ID,} from "@solana/spl-token"; // IGNORE THESE ERRORS IF ANYimport { SolanaBlockchain } from "../target/types/solana\_blockchain";const { SystemProgram, PublicKey } = anchor.web3;describe("metaplex-anchor-nft", () => { // Configure the client to use the local cluster. const provider = anchor.AnchorProvider.env(); const wallet = provider.wallet as Wallet; anchor.setProvider(provider); const program = anchor.workspace .SolanaBlockchain as Program<SolanaBlockchain>; //TODO: Check whether if the creator key has to match with the mint key const mintKey: anchor.web3.Keypair = anchor.web3.Keypair.generate(); it("creates a new mint account!", async () => { //create a new keypair for a new mint account const lamports: number = await program.provider.connection.getMinimumBalanceForRentExemption( MINT\_SIZE ); console.log(`mintKey address is ${mintKey.publicKey}`); const NftTokenAccount = await getAssociatedTokenAddress( mintKey.publicKey, wallet.publicKey ); console.log("NFT Account: ", NftTokenAccount.toBase58()); const mint\_tx = new anchor.web3.Transaction().add( anchor.web3.SystemProgram.createAccount({ fromPubkey: wallet.publicKey, newAccountPubkey: mintKey.publicKey, space: MINT\_SIZE, programId: TOKEN\_PROGRAM\_ID, lamports, }), createInitializeMintInstruction( mintKey.publicKey, 0, wallet.publicKey, wallet.publicKey ), createAssociatedTokenAccountInstruction( wallet.publicKey, NftTokenAccount, wallet.publicKey, mintKey.publicKey ) ); const res = await program.provider.sendAndConfirm(mint\_tx, [mintKey]); console.log( await program.provider.connection.getParsedAccountInfo(mintKey.publicKey) ); console.log("Account: ", res); console.log("Mint key: ", mintKey.publicKey.toString()); console.log("User: ", wallet.publicKey.toString()); const current\_supply\_before = parseInt( (await provider.connection.getTokenSupply(mintKey.publicKey)).value.amount ); console.log(`current supply is ${current\_supply\_before}`); assert.ok(current\_supply\_before === 0); const tx = await program.methods .mintNft() .accounts({ mint: mintKey.publicKey, tokenAccount: NftTokenAccount, tokenProgram: TOKEN\_PROGRAM\_ID, payer: wallet.publicKey, }) .rpc(); console.log("Your transaction signature", tx); const current\_supply\_after = parseInt( (await provider.connection.getTokenSupply(mintKey.publicKey)).value.amount ); console.log(`current supply is ${current\_supply\_after}`); assert.ok(current\_supply\_after === 1); }); it("Is creates a new metadata account!", async () => { // Add your test here. const TOKEN\_METADATA\_PROGRAM\_ID = new anchor.web3.PublicKey( "metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s" ); const getMetadata = async ( mint: anchor.web3.PublicKey ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const metadataAddress = await getMetadata(mintKey.publicKey); console.log("Metadata address: ", metadataAddress.toBase58()); const tx = await program.methods .createMetadataAccount( mintKey.publicKey, "https://gateway.pinata.cloud/ipfs/QmQNtiFGzdo8eQbmHTGpxX3LwhmWe49StstvLqXx8GRt3E", "Ronnie Coleman NFT", "LWB" ) .accounts({ mintAuthority: wallet.publicKey, mint: mintKey.publicKey, tokenProgram: TOKEN\_PROGRAM\_ID, metadata: metadataAddress, tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, payer: wallet.publicKey, systemProgram: SystemProgram.programId, rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, }) .rpc(); console.log("Your transaction signature", tx); }); it("Is creates a new master account!", async () => { // Add your test here. const TOKEN\_METADATA\_PROGRAM\_ID = new anchor.web3.PublicKey( "metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s" ); const getMetadata = async ( mint: anchor.web3.PublicKey ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const getMasterEdition = async ( mint: anchor.web3.PublicKey ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), Buffer.from("edition"), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const metadataAddress = await getMetadata(mintKey.publicKey); const masterEdition = await getMasterEdition(mintKey.publicKey); // console.log("Metadata address: ", metadataAddress.toBase58()); console.log("MasterEdition: ", masterEdition.toBase58()); const tx = await program.methods .createMasterEdition(null) .accounts({ mintAuthority: wallet.publicKey, mint: mintKey.publicKey, tokenProgram: TOKEN\_PROGRAM\_ID, metadata: metadataAddress, tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, payer: wallet.publicKey, systemProgram: SystemProgram.programId, rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, masterEdition: masterEdition, }) .rpc(); console.log("Your transaction signature", tx); }); it("creates a new mint edition nft from the original master edition nft", async () => { //generate a new mint account and a new associated token account const new\_mintKey = anchor.web3.Keypair.generate(); const lamports: number = await program.provider.connection.getMinimumBalanceForRentExemption( MINT\_SIZE ); console.log(`a new mintKey address is ${new\_mintKey.publicKey}`); const NewNftTokenAccount = await getAssociatedTokenAddress( new\_mintKey.publicKey, wallet.publicKey ); console.log("The New NFT Account: ", NewNftTokenAccount.toBase58()); const mint\_tx = new anchor.web3.Transaction().add( anchor.web3.SystemProgram.createAccount({ fromPubkey: wallet.publicKey, newAccountPubkey: new\_mintKey.publicKey, space: MINT\_SIZE, programId: TOKEN\_PROGRAM\_ID, lamports, }), createInitializeMintInstruction( new\_mintKey.publicKey, 0, wallet.publicKey, wallet.publicKey ), createAssociatedTokenAccountInstruction( wallet.publicKey, NewNftTokenAccount, wallet.publicKey, new\_mintKey.publicKey ) ); const res = await program.provider.sendAndConfirm(mint\_tx, [new\_mintKey]); console.log( await program.provider.connection.getParsedAccountInfo( new\_mintKey.publicKey ) ); console.log("Account: ", res); console.log("New Mint key: ", new\_mintKey.publicKey.toString()); console.log("User: ", wallet.publicKey.toString()); const current\_supply\_before = parseInt( (await provider.connection.getTokenSupply(new\_mintKey.publicKey)).value .amount ); console.log(`current supply is ${current\_supply\_before}`); assert.ok(current\_supply\_before === 0); const tx = await program.methods .mintNft() .accounts({ mint: new\_mintKey.publicKey, tokenAccount: NewNftTokenAccount, tokenProgram: TOKEN\_PROGRAM\_ID, payer: wallet.publicKey, }) .rpc(); console.log("Your transaction signature", tx); const current\_supply\_after = parseInt( (await provider.connection.getTokenSupply(new\_mintKey.publicKey)).value .amount ); console.log(`current supply is ${current\_supply\_after}`); assert.ok(current\_supply\_after === 1); console.log( `a new mint account has been created at ${new\_mintKey.publicKey}` ); //TODO: get the metadata and master edition, as well as the token account of the original mint account const TOKEN\_METADATA\_PROGRAM\_ID = new anchor.web3.PublicKey( "metaqbxxUerdq28cj1RbAWkYQm3ybzjb6a8bt518x1s" ); const NftTokenAccount = await getAssociatedTokenAddress( mintKey.publicKey, wallet.publicKey ); const getMetadata = async ( mint: anchor.web3.PublicKey ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const metadataAddress = await getMetadata(mintKey.publicKey); console.log("Metadata address: ", metadataAddress.toBase58()); // const tx1 = await program.methods // .createMetadataAccount( // new\_mintKey.publicKey, // "https://gateway.pinata.cloud/ipfs/QmQNtiFGzdo8eQbmHTGpxX3LwhmWe49StstvLqXx8GRt3E", // "Ronnie Coleman NFT", // "LWB" // ) // .accounts({ // mintAuthority: wallet.publicKey, // mint: mintKey.publicKey, // tokenProgram: TOKEN\_PROGRAM\_ID, // metadata: metadataAddress, // tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, // payer: wallet.publicKey, // systemProgram: SystemProgram.programId, // rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, // }) // .rpc(); // console.log("Your transaction signature", tx1); const getMasterEdition = async ( mint: anchor.web3.PublicKey ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), Buffer.from("edition"), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const masterEdition = await getMasterEdition(mintKey.publicKey); console.log("MasterEdition: ", masterEdition.toBase58()); // const tx2 = await program.methods // .createMasterEdition(null) // .accounts({ // mintAuthority: wallet.publicKey, // mint: mintKey.publicKey, // tokenProgram: TOKEN\_PROGRAM\_ID, // metadata: metadataAddress, // tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, // payer: wallet.publicKey, // systemProgram: SystemProgram.programId, // rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, // masterEdition: masterEdition, // }) // .rpc(); // console.log("Your transaction signature", tx2); //TODO: Create a new metadata account const newMetadataAddress = await getMetadata(new\_mintKey.publicKey); console.log("New metadata address: ", newMetadataAddress.toBase58()); const tx3 = await program.methods .createMetadataAccount( new\_mintKey.publicKey, "https://gateway.pinata.cloud/ipfs/QmQNtiFGzdo8eQbmHTGpxX3LwhmWe49StstvLqXx8GRt3E", "Ronnie Coleman NFT", "LWB" ) .accounts({ mintAuthority: wallet.publicKey, mint: new\_mintKey.publicKey, tokenProgram: TOKEN\_PROGRAM\_ID, metadata: newMetadataAddress, tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, payer: wallet.publicKey, systemProgram: SystemProgram.programId, rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, }) .rpc(); console.log("Your transaction signature", tx3); //TODO: Create a new edition account const newMasterEdition = await getMasterEdition(new\_mintKey.publicKey); console.log("New masterEdition: ", newMasterEdition.toBase58()); const tx4 = await program.methods .createMasterEdition(null) .accounts({ mintAuthority: wallet.publicKey, mint: new\_mintKey.publicKey, tokenProgram: TOKEN\_PROGRAM\_ID, metadata: newMetadataAddress, tokenMetadataProgram: TOKEN\_METADATA\_PROGRAM\_ID, payer: wallet.publicKey, systemProgram: SystemProgram.programId, rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, masterEdition: newMasterEdition, }) .rpc(); console.log("Your transaction signature", tx4); //TODO: CREATE a Edition Mark PDA const getEditionMarkPDA = async ( mint: anchor.web3.PublicKey, edition\_number: number ): Promise<anchor.web3.PublicKey> => { return ( await anchor.web3.PublicKey.findProgramAddress( [ Buffer.from("metadata"), TOKEN\_METADATA\_PROGRAM\_ID.toBuffer(), mint.toBuffer(), Buffer.from("edition"), new Uint8Array(edition\_number / 248), ], TOKEN\_METADATA\_PROGRAM\_ID ) )[0]; }; const editionMarkPDA = await getEditionMarkPDA(mintKey.publicKey, 1); console.log(`EditionMarkPDA address is ${editionMarkPDA}`); const tx5 = await program.methods .createNewEditionNft((1)[0]) .accounts({ originalMint: mintKey.publicKey, newMetadata: newMetadataAddress, newEdition: newMasterEdition, masterEdition: masterEdition, newMint: new\_mintKey.publicKey, editionMarkPda: editionMarkPDA, newMintAuthority: wallet.publicKey, payer: wallet.publicKey, tokenAccountOwner: wallet.publicKey, tokenAccount: NftTokenAccount, newMetadataUpdateAuthority: wallet.publicKey, metadata: metadataAddress, tokenProgram: TOKEN\_PROGRAM\_ID, systemProgram: SystemProgram.programId, rent: anchor.web3.SYSVAR\_RENT\_PUBKEY, }) .rpc(); console.log("Your transaction signature", tx5); });});  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Please let me know if you can tell me what am I missing or doing wrong here :(  
  
Thanks!

**C1:**Did you find a solution? I am facing the same problem.

**C2:**Stuck here as well, any solutions?

1 **Answer**

**A1:**I have not run your code but it seems like the accounts you are passing are incorrect.  
  
 ● create\_metadata\_account:  
  
The list of accounts can be found here  
You are passing token\_program while it's not needed.  
update\_authority\_info is missing  
  
  
  
 ● create\_master\_edition:  
  
The list of accounts can be found here  
update\_authority\_info is missing  
  
  
  
 ● create\_new\_edition\_nft:  
  
The list of accounts can be found here  
update\_authority\_info is missing  
token\_program\_account\_info is missing

**C1:**Hey, thank you for answering. Btw, I think I did pass update\_authority\_info and token\_program\_account\_info (you can check the above code. I passed those values in newMetadataUpdateAuthority and tokenProgram field). Am I still missing something or misunderstood your suggestion? Lmk!