**Debug your smart contract**

**SVG NFT Debugging**

In the last lesson we left off with a gross error that hit us when running our new integration test testFlipMoodIntegration. Let's run the test again with the verbose flag and see if we can debug what's going on.

forge test --mt testFlipMoodIntegration -vvv

Hmm, this gives us a little more information, detailing that our assertion failed as well as providing us an output of one of the SVG URIs, but I think we can do better.

This is where I like to employ assertEq instead of assert as this will print both the left and right sides of the assertion to our console.

assertEq(

keccak256(abi.encodePacked(moodNft.tokenURI(0))),

keccak256(abi.encodePacked(SAD\_SVG\_URI))

);

Let's run it again.

forge test --mt testFlipMoodIntegration -vvv

Well, our hashes are definitely different. We can import console and log out some variables to see what's going wrong.

import {console, Test} from "forge-std/Test.sol";

...

contract MoodNFTTest is Test {

...

function testFlipMoodIntegration() public {

vm.prank(USER);

moodNFT.mintNft();

vm.prank(USER);

moodNFT.flipMood(0);

console.log(moodNFT.tokenURI(0))

assertEq(keccak256(abi.encodePacked(moodNFT.tokenURI(0))), keccak256(abi.encodePacked(SAD\_SVG\_URI)));

}

}

Running this now, should output our tokenURI, which we can verify in our browser, we expect the Sad SVG, so what do we get?

├─ [0] console::log("data:application/json;base64,") [staticcall]

Pasting this into our browser and checking the imageUri we should be able to verify that this *is* the Sad tokenUri.. so what's going on?

Let's check the other side of the assertion. We have the SAD\_SVG\_URI as a constant variable, let's toss it into our browser.

Wait a minute! One of these is returning our ***tokenURI*** and the other is our ***imageURI***! This is why it's important to be explicit in our naming conventions! Let's adjust these constants, and our test, right away. We can define a variable with what we expect the ***tokenURI*** to be and assert versus that.

string public constant SAD\_SVG\_IMAGE\_URI = ...;

string public constant HAPPY\_SVG\_IMAGE\_URI = ...;

string public constant SAD\_SVG\_URI = "data:application/json;base64,"

...

function testFlipMoodIntegration() public {

vm.prank(USER);

moodNFT.mintNft();

vm.prank(USER);

moodNFT.flipMood(0);

assertEq(keccak256(abi.encodePacked(moodNFT.tokenURI(0))), keccak256(abi.encodePacked(SAD\_SVG\_URI)));

}

With these adjustments, we can run our test again...

Beautiful!

**Wrap Up**

Ok, we've done a lot. We've structured our test suite such that we now leverage integration tests that are using our DeployMoodNft.s.sol script.

We're testing minting, SVG encoding, deploying and more. One thing we didn't do together was an interations script for our MoodNft contract. In practice this should be very similar to what we've written for our BasicNFT in Interactions.s.sol so far.

I highly encourage you to try to write this script for MoodNFT. It should be able to mint the NFT and flip the mood!

Your second call to action is going to be increasing the coverage of our contracts. Write some tests and try to get MoodNFT and our scripts closer to 100%!