**Building an NFT Auction Smart Contract on Ethereum**

[[](https://medium.com/@christianinyekaka?source=post_page-----4a1be8795283--------------------------------)](https://medium.com/@christianinyekaka?source=post_page-----4a1be8795283--------------------------------)

[[CoinsBench](https://coinsbench.com/?source=post_page-----4a1be8795283--------------------------------)](https://coinsbench.com/?source=post_page-----4a1be8795283--------------------------------)

[Christianinyekaka](https://medium.com/@christianinyekaka?source=post_page-----4a1be8795283--------------------------------)

·

Follow

Published in

[CoinsBench](https://coinsbench.com/?source=post_page-----4a1be8795283--------------------------------)

·

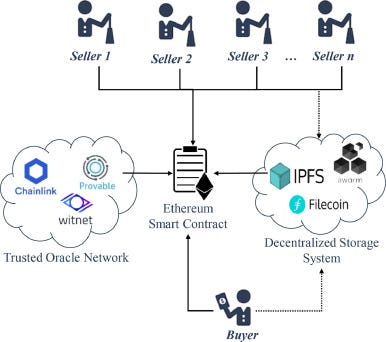
11 min read

·

Dec 31, 2023

15

2



Auction Smart Contract

The decentralized nature of blockchain technology has opened up innovative possibilities for digital assets, notably in the form of Non-Fungible Tokens (NFTs). NFTs represent unique digital assets, and auctions have become a popular way to trade them. In this tutorial, we’ll explore how to build an Ethereum-based NFT auction smart contract.

**Prerequisites:**

* Basic understanding of Solidity.
* Development environment setup with tools like Hardhat or Truffle.

**Smart Contract Structure**

// SPDX-License-Identifier: MIT  
  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";  
// import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Pausable.sol";  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol";  
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
contract MyToken is ERC721, ERC721Enumerable, ERC721URIStorage, ERC721Burnable, Ownable {  
 constructor(address initialOwner)  
 ERC721("MyToken", "MTK")  
 Ownable(initialOwner)  
 {}  
  
 function safeMint(address to, uint256 tokenId, string memory uri)  
 public  
 onlyOwner   
 {  
 \_safeMint(to, tokenId);  
 \_setTokenURI(tokenId, uri);  
 }  
 // function pause() external onlyOwner {  
 // \_pause();  
 // }  
 // function unpause() public onlyOwner {  
 // \_unpause();  
 // }  
  
 // The following functions are overrides required by Solidity.  
  
 function \_update(address to, uint256 tokenId, address auth)  
 internal  
 override(ERC721, ERC721Enumerable)  
 returns (address)  
 {  
 return super.\_update(to, tokenId, auth);  
 }  
  
 function \_increaseBalance(address account, uint128 value)  
 internal  
 override(ERC721, ERC721Enumerable)  
 {  
 super.\_increaseBalance(account, value);  
 }  
  
 function tokenURI(uint256 tokenId)  
 public  
 view  
 override(ERC721, ERC721URIStorage)  
 returns (string memory)  
 {  
 return super.tokenURI(tokenId);  
 }  
  
 function supportsInterface(bytes4 interfaceId)  
 public  
 view  
 override(ERC721, ERC721Enumerable, ERC721URIStorage)  
 returns (bool)  
 {  
 return super.supportsInterface(interfaceId);  
 }  
   
}  
contract Auction is ReentrancyGuard {  
 struct AuctionInfo {  
 address seller;  
 uint256 tokenId;  
 uint256 highestBid;  
 address payable highestBidder;  
 uint256 endTimestamp;  
 bool ended;  
 }  
  
 mapping(uint256 => AuctionInfo) public auctions;  
 mapping(uint256 => bool) public claimedNFT;  
  
 event NewBid(address indexed bidder, uint256 indexed tokenId, uint256 amount);  
 event AuctionEnded(uint256 indexed tokenId, address indexed winner, uint256 amount);  
 event Claimed(address indexed bidder, uint256 indexed tokenID);  
 event moneySent(address indexed seller , uint256 indexed tokenID, uint256 amount);  
  
 modifier auctionExists(uint256 tokenId) {  
 require(auctions[tokenId].seller != address(0), "Auction does not exist");  
 \_;  
 }  
  
 modifier onlySeller(uint256 tokenId) {  
 require(auctions[tokenId].seller == msg.sender, "Not the auction seller");  
 \_;  
 }  
  
 constructor() {  
   
 }  
 // this function return the sellers address  
 function getSeller(uint256 auctionId) external view returns (address) {  
 return auctions[auctionId].seller;  
 }  
 // this function return the highest bidder address  
 function getHighestBidder(uint256 auctionId) external view returns (address) {  
 return auctions[auctionId].highestBidder;  
 }  
 function isEnded(uint256 auctionId) external view returns (bool) {  
 return auctions[auctionId].ended;  
 }  
// this function start the auction  
 function startAuction(  
 address tokenContract,  
 uint256 tokenId,  
 uint256 duration  
 ) external nonReentrant {  
 MyToken nft = MyToken(tokenContract);  
 require(nft.ownerOf(tokenId) == msg.sender, "Not token owner");  
 require(duration > 0, "Duration should be > 0");  
  
 nft.transferFrom(msg.sender, address(this), tokenId);  
  
 auctions[tokenId] = AuctionInfo({  
 seller: msg.sender,  
 tokenId: tokenId,  
 highestBid: 0,  
 highestBidder: payable(address(0)),  
 endTimestamp: block.timestamp + duration,  
 ended: false  
 });  
 }  
 // this function post a bid and replace the highest bid if the new bid is higher than the highest bid  
  
 function bid(uint256 tokenId) external payable nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
 require(block.timestamp < auction.endTimestamp, "Auction ended");  
 require(msg.value > auction.highestBid, "Bid too low");  
  
 if (auction.highestBidder != address(0)) {  
 auction.highestBidder.transfer(auction.highestBid);  
 }  
  
 auction.highestBid = msg.value;  
 auction.highestBidder = payable(msg.sender);  
 emit NewBid(msg.sender, tokenId, msg.value);  
 }  
 // this function ends the auction  
  
 function endAuction(uint256 tokenId) external nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
 require(block.timestamp >= auction.endTimestamp, "Auction not ended");  
 require(!auction.ended, "Auction already ended");  
 auction.ended = true;  
  
 emit AuctionEnded(tokenId, auction.highestBidder, auction.highestBid);  
 }  
 // this function claims the NFT and thransfer the nft to the winner and transfer the money to the seller  
 function claimNFT(uint256 tokenId,address tokenContract) external nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
  
 require(auction.ended, "Auction not ended");  
  
 require(auction.highestBidder == msg.sender,"not the winner");  
  
 MyToken nft = MyToken(tokenContract);  
  
 if(!claimedNFT[tokenId]){  
  
 nft.transferFrom(address(this), msg.sender, tokenId);  
  
 claimedNFT[tokenId] = true;   
  
 payable(auction.seller).transfer(auction.highestBid);   
  
 emit Claimed(msg.sender, tokenId);  
  
 emit moneySent(auction.seller, tokenId, auction.highestBid);  
  
 }else{  
 revert("already claimed");  
 }  
   
 }  
// this function withdraw the NFT if no winner or the bid from the winner  
 function withdraw(uint256 tokenId,address tokenContract) external nonReentrant auctionExists(tokenId) {  
  
 AuctionInfo storage auction = auctions[tokenId];  
  
 require(auction.ended, "Auction not ended");  
  
 if (msg.sender == auction.seller) {  
  
 require(auction.highestBidder == address(0), "Cannot withdraw, auction ended");  
  
 MyToken nft = MyToken(tokenContract);  
  
 // tranfer the nft back to the seller  
 nft.transferFrom(address(this), auction.seller, tokenId);  
  
 } else if (msg.sender == auction.highestBidder) {  
  
 require(!claimedNFT[tokenId], "Bidder already received NFT");  
  
 payable(msg.sender).transfer(auction.highestBid);  
  
 } else {  
 revert("Not allowed to withdraw");  
 }  
 }  
  
}

The smart contract is composed of two key contracts: MyToken and Auction. MyToken extends ERC721, ERC721Enumerable, ERC721URIStorage, ERC721Burnable, Ownable. Facilitating the creation and management of NFTs. On the other hand, Auction extends **ReentrancyGuard.**The ReentrancyGuard is a protective mechanism used in Solidity to prevent reentrant attacks in smart contracts A reentrancy attack occurs when an external attacker exploits a contract's vulnerable state by repeatedly calling back into the contract before the ongoing execution completes. ReentrancyGuard is a crucial security measure in Solidity contracts, particularly in scenarios involving asset transfers, to prevent reentrancy attacks and ensure the integrity of contract state changes during critical operations. manages the auction process for these NFTs, governing bidding and ownership transfer.

**Setting Up The NFT:**

contract MyToken is ERC721, ERC721Enumerable, ERC721URIStorage, ERC721Burnable, Ownable {  
 constructor(address initialOwner)  
 ERC721("MyToken", "MTK")  
 Ownable(initialOwner)  
 {}  
  
 function safeMint(address to, uint256 tokenId, string memory uri)  
 public  
 onlyOwner   
 {  
 \_safeMint(to, tokenId);  
 \_setTokenURI(tokenId, uri);  
 }  
 // function pause() external onlyOwner {  
 // \_pause();  
 // }  
 // function unpause() public onlyOwner {  
 // \_unpause();  
 // }  
  
 // The following functions are overrides required by Solidity.  
  
 function \_update(address to, uint256 tokenId, address auth)  
 internal  
 override(ERC721, ERC721Enumerable)  
 returns (address)  
 {  
 return super.\_update(to, tokenId, auth);  
 }  
  
 function \_increaseBalance(address account, uint128 value)  
 internal  
 override(ERC721, ERC721Enumerable)  
 {  
 super.\_increaseBalance(account, value);  
 }  
  
 function tokenURI(uint256 tokenId)  
 public  
 view  
 override(ERC721, ERC721URIStorage)  
 returns (string memory)  
 {  
 return super.tokenURI(tokenId);  
 }  
  
 function supportsInterface(bytes4 interfaceId)  
 public  
 view  
 override(ERC721, ERC721Enumerable, ERC721URIStorage)  
 returns (bool)  
 {  
 return super.supportsInterface(interfaceId);  
 }  
   
}

Function safeMint is used to mint ERC721 tokens safely. It can only be called by the owner of the contract (onlyOwner modifier). It takes \_to an address tokenIdand a uri string as arguments. Inside the function, it mints a new token to the \_to address and sets its URI using \_setTokenURI.

Function tokenURI overrides the default implementation in ERC721. It retrieves and returns the URI associated with a given \_tokenId. It ensures that the token exists before querying its URI.

Function supportsInterface overrides the function from ERC721 and ERC721Enumerable contracts. It checks whether a particular interface is supported by the contract by calling the supportsInterface function from the inherited contracts.

**Auction Contract Overview:**

**Auction Struct:**

* To encapsulate auction details effectively, a `struct` is utilized, capturing essential information such as the seller, the TokenID, the highest bid, the highest bidder, and timestamp, and the ended state.

struct AuctionInfo {  
 address seller;  
 uint256 tokenId;  
 uint256 highestBid;  
 address payable highestBidder;  
 uint256 endTimestamp;  
 bool ended;  
 }

**Mapping and Variables:**

* We create mappings to store auctions for each token ID and the claimed NFTs.

mapping(uint256 => AuctionInfo) public auctions;  
mapping(uint256 => bool) public claimedNFT;

**Events:**

These events serve to notify external systems or users about critical actions taking place within the auction contract, allowing them to track the auction’s progress, facilitate user interface updates, or trigger further actions in response to these events.

event NewBid(address indexed bidder, uint256 indexed tokenId, uint256 amount);  
 event AuctionEnded(uint256 indexed tokenId, address indexed winner, uint256 amount);  
 event Claimed(address indexed bidder, uint256 indexed tokenID);  
 event moneySent(address indexed seller , uint256 indexed tokenID, uint256 amount);

* NewBid: Signals that a new bid has been placed, providing information about the bidder, the token ID, and the bid amount.
* AuctionEnded: Indicates that an auction has concluded, specifying the token ID, the winning bidder, and the winning bid amount.
* Claimed: Signals that a bidder has claimed an NFT, providing information about the bidder and the token ID.
* moneySent: Indicates that money has been transferred, specifying the seller's address, the token ID, and the amount sent.

**Start Auction:**

Initiates a new auction by transferring the NFT to the contract and defining auction parameters.

function startAuction(  
 address tokenContract,  
 uint256 tokenId,  
 uint256 duration  
 ) external nonReentrant {  
 MyToken nft = MyToken(tokenContract);  
 require(nft.ownerOf(tokenId) == msg.sender, "Not token owner");  
 require(duration > 0, "Duration should be > 0");  
  
 nft.transferFrom(msg.sender, address(this), tokenId);  
  
 auctions[tokenId] = AuctionInfo({  
 seller: msg.sender,  
 tokenId: tokenId,  
 highestBid: 0,  
 highestBidder: payable(address(0)),  
 endTimestamp: block.timestamp + duration,  
 ended: false  
 });  
 }

**Place Bid:**

Allows users to bid on ongoing auctions, updating the highest bid and bidder.

function bid(uint256 tokenId) external payable nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
 require(block.timestamp < auction.endTimestamp, "Auction ended");  
 require(msg.value > auction.highestBid, "Bid too low");  
  
 if (auction.highestBidder != address(0)) {  
 auction.highestBidder.transfer(auction.highestBid);  
 }  
  
 auction.highestBid = msg.value;  
 auction.highestBidder = payable(msg.sender);  
 emit NewBid(msg.sender, tokenId, msg.value);  
 }

**End Auction:**

Ends the auction after a specified duration and emits the AuctionEnded event

function endAuction(uint256 tokenId) external nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
 require(block.timestamp >= auction.endTimestamp, "Auction not ended");  
 require(!auction.ended, "Auction already ended");  
 auction.ended = true;  
  
 emit AuctionEnded(tokenId, auction.highestBidder, auction.highestBid);  
 }

**Claim NFT:**

Enables the winner to claim the NFT and facilitates fund transfer to the seller.

function claimNFT(uint256 tokenId,address tokenContract) external nonReentrant auctionExists(tokenId) {  
 AuctionInfo storage auction = auctions[tokenId];  
  
 require(auction.ended, "Auction not ended");  
  
 require(auction.highestBidder == msg.sender,"not the winner");  
  
 MyToken nft = MyToken(tokenContract);  
  
 if(!claimedNFT[tokenId]){  
  
 nft.transferFrom(address(this), msg.sender, tokenId);  
  
 claimedNFT[tokenId] = true;   
  
 payable(auction.seller).transfer(auction.highestBid);   
  
 emit Claimed(msg.sender, tokenId);  
  
 emit moneySent(auction.seller, tokenId, auction.highestBid);  
  
 }else{  
 revert("already claimed");  
 }  
   
 }

**Withdraw NFT:**

Allows the seller to withdraw the NFT if there are no bids or if the auction ended unsuccessfully. if the auction winner did not claim their NFT or if they don’t want the NFT anymore.

function withdraw(uint256 tokenId,address tokenContract) external nonReentrant auctionExists(tokenId) {  
  
 AuctionInfo storage auction = auctions[tokenId];  
  
 require(auction.ended, "Auction not ended");  
  
 if (msg.sender == auction.seller) {  
  
 require(auction.highestBidder == address(0), "Cannot withdraw, auction ended");  
  
 MyToken nft = MyToken(tokenContract);  
  
 // tranfer the nft back to the seller  
 nft.transferFrom(address(this), auction.seller, tokenId);  
  
 } else if (msg.sender == auction.highestBidder) {  
  
 require(!claimedNFT[tokenId], "Bidder already received NFT");  
  
 payable(msg.sender).transfer(auction.highestBid);  
  
 } else {  
 revert("Not allowed to withdraw");  
 }  
 }

**Modifiers:**

modifier auctionExists(uint256 tokenId) {  
 require(auctions[tokenId].seller != address(0), "Auction does not exist");  
 \_;  
 }  
  
 modifier onlySeller(uint256 tokenId) {  
 require(auctions[tokenId].seller == msg.sender, "Not the auction seller");  
 \_;  
 }

The modifiers auctionExists and onlySeller play a crucial role in ensuring certain conditions are met before executing functions in the smart contract.

1. auctionExists(uint256 tokenId): This modifier ensures that an auction exists for a given token ID before allowing the execution of functions that use this modifier. It checks if an auction exists by verifying that the seller's address associated with the given token ID is not the null address (address(0)). If the auction does not exist, it prevents the function from proceeding and reverts with the message "Auction does not exist."
2. onlySeller(uint256 tokenId): This modifier restricts the execution of functions to only the seller of a specific auction identified by the provided token ID. It ensures that only the seller (the address that initiated the auction) can execute functions with this modifier. If the caller is not the auction seller, the function will not proceed and will revert with the message "Not the auction seller."

These modifiers serve as preconditions before executing specific functions within the contract, ensuring that only authorized individuals (such as the seller or when an auction exists) can interact with certain functionalities, enhancing security and preventing unauthorized access or manipulation of auction-related operations.

**Additional Functions:**

// this function return the sellers address  
 function getSeller(uint256 auctionId) external view returns (address) {  
 return auctions[auctionId].seller;  
 }  
 // this function return the highest bidder address  
 function getHighestBidder(uint256 auctionId) external view returns (address) {  
 return auctions[auctionId].highestBidder;  
 }  
 function isEnded(uint256 auctionId) external view returns (bool) {  
 return auctions[auctionId].ended;  
 }

These additional functions within the smart contract further enhance its functionality and transparency:

1. getSeller(uint256 auctionId): This function, when called externally, retrieves and returns the address of the seller associated with a specific auction ID. It provides transparency by allowing anyone to query and verify the seller's address for a given auction.
2. getHighestBidder(uint256 auctionId): Similar to the getSeller function, this function externally retrieves and returns the address of the highest bidder for a particular auction ID. It offers visibility into the current highest bidder in an auction.
3. isEnded(uint256 auctionId): This function, when called externally, checks and returns a boolean value indicating whether a specific auction has ended. It's useful for external parties to determine the status of an auction—whether it's ongoing or has already concluded.

These functions essentially provide a read-only interface to certain key details of the auctions, empowering users or external systems to query the contract for critical information such as seller details, highest bidder, and auction status without altering the contract’s state. This transparency and accessibility contribute to the overall usability and trustworthiness of the auction smart contract.

**Testing and Deployment:**

To ensure the contract’s functionality and security, comprehensive testing using tools like Hardhat or Truffle is essential. Deploying the contract to a testnet or a local development environment provides a safe space to evaluate its behavior before deploying it to the Ethereum mainnet.

**Test:**

const { expect } = require("chai");  
const { ethers } = require("hardhat");  
const { network } = require("hardhat");  
  
describe("Auction", function () {  
 let myToken;  
 let owner;  
 let addr1;  
 let addr2;  
 let auction;  
 const amount = ethers.parseEther("1");  
 const lowerAmount = ethers.parseEther("0.5");  
 const higherAmount = ethers.parseEther("1.5");  
  
 beforeEach(async () => {  
 [owner, addr1, addr2] = await ethers.getSigners();  
  
 const MyToken = await ethers.getContractFactory("MyToken");  
 myToken = await MyToken.deploy(owner.address);  
 await myToken.waitForDeployment();  
  
 const Auction = await ethers.getContractFactory("Auction");  
 auction = await Auction.deploy();  
 await auction.waitForDeployment();  
 });  
  
 it("should deploy", async function () {  
 expect(await myToken.owner()).to.equal(owner.address);  
 expect(await auction.target).to.not.equal(undefined);  
 });  
  
 it("should mint an NFT", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
 const ownerOfToken = await myToken.ownerOf(2);  
 expect(ownerOfToken).to.equal(owner.address);  
 });  
  
 it("should create auction for an NFT", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 300000000);  
  
 const seller = await Contract.getSeller(2);  
 expect(seller).to.equal(owner.address);  
 });  
 it("should bid on an auction", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 300000000);  
  
 await Contract.connect(addr1).bid(2, { value: amount });  
  
 const highestBidder = await Contract.getHighestBidder(2);  
 expect(highestBidder).to.equal(addr1.address);  
 });  
 it("should not bid on an auction if bid is lower than highest bid", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 300000000);  
  
 await Contract.connect(addr1).bid(2, { value: amount });  
 await expect(  
 Contract.connect(addr2).bid(2, { value: lowerAmount })  
 ).to.be.revertedWith("Bid too low");  
 });  
 it("should change the highest bid", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 300000000);  
  
 await Contract.connect(addr1).bid(2, { value: amount });  
 await Contract.connect(addr2).bid(2, { value: higherAmount });  
  
 const highestBidder = await Contract.getHighestBidder(2);  
 expect(highestBidder).to.equal(addr2.address);  
 });  
 it("should transfer back the bid in case of a higher bid", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 300000000);  
  
 await Contract.connect(addr1).bid(2, { value: amount });  
 const blance = await ethers.provider.getBalance(addr1.address);  
 await Contract.connect(addr2).bid(2, { value: higherAmount });  
  
 const balance = await ethers.provider.getBalance(addr1.address);  
  
 expect(balance - blance).to.equal(amount);  
 });  
 it("should end an auction", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 3600);  
 await Contract.connect(addr1).bid(2, { value: amount });  
 await network.provider.request({  
 method: "evm\_increaseTime",  
 params: [3601],  
 });  
 await Contract.connect(owner).endAuction(2);  
  
 const isended = await Contract.isEnded(2);  
  
 expect(isended).to.equal(true);  
 });  
  
 it("should allow the highest bidder to claim the NFT", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 3600);  
 await Contract.connect(addr1).bid(2, { value: amount });  
 await network.provider.request({  
 method: "evm\_increaseTime",  
 params: [3601],  
 });  
 await Contract.connect(owner).endAuction(2);  
 await Contract.connect(addr1).claimNFT(2, myToken.target);  
 const ownerOfToken = await myToken.ownerOf(2);  
 expect(ownerOfToken).to.equal(addr1.address);  
 });  
 it("should allow the highest bidder to withdraw", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 3600);  
 await Contract.connect(addr1).bid(2, { value: amount });  
 const blance = await ethers.provider.getBalance(addr1.address);  
 await network.provider.request({  
 method: "evm\_increaseTime",  
 params: [3601],  
 });  
 await Contract.connect(owner).endAuction(2);  
  
 await Contract.connect(addr1).withdraw(2, myToken.target);  
 const balance = await ethers.provider.getBalance(addr1.address);  
  
 expect(balance).to.be.greaterThan(blance);  
 });  
 it("should withdraw NFT", async function () {  
 await myToken.safeMint(  
 owner.address,  
 2,  
 "ipfs://bafkreia3qm2s35beymhuma7w4x3bp3bn45vamm532lgwzi5vdv3sbv4sae"  
 );  
  
 const Contract = await ethers.getContractAt("Auction", auction.target);  
 await myToken.approve(auction.target, 2);  
 await Contract.connect(owner).startAuction(myToken.target, 2, 3600);  
 await network.provider.request({  
 method: "evm\_increaseTime",  
 params: [3601],  
 });  
 await Contract.connect(owner).endAuction(2);  
 const isended = await Contract.isEnded(2);  
 await Contract.connect(owner).withdraw(2, myToken.target);  
 // check gas used  
  
 expect(await myToken.ownerOf(2)).to.equal(owner.address);  
 });  
});

**Conclusion**

Building an NFT auction smart contract involves integrating ERC721 functionalities for NFT management and implementing auction-related features for bidding and ownership transfer. Thorough testing and proper deployment are crucial to ensure the contract’s reliability and security.

**Final Thoughts**

Developing an NFT auction on Ethereum involves integrating ERC721 standards for NFT management and creating auction-specific functionalities. Security measures like ReentrancyGuard are crucial to prevent attacks during asset transfers. Thorough testing using tools like Hardhat or Truffle and deploying contracts to testnets ensure functionality and security before the Ethereum mainnet launch. Embracing blockchain's decentralization and following Solidity best practices contribute to DeFi's growth and wider NFT adoption, unlocking new possibilities in the digital asset realm

**Community and Resources**

1. Online Communities: Engage with blockchain developer communities, forums, and platforms like Ethereum Stack Exchange, Reddit’s r/ethereum, or Discord/Telegram groups.
2. Documentation and Tutorials: Refer to official Ethereum documentation, Solidity documentation, QuickNode, Alchemy University courses, and various tutorials available online.

*This marks the end of our tutorial if you found it useful feel free to clap or leave a comment*. *Any suggestions or feedback will be welcomed. You can also connect with me on* [***Twitter***](https://twitter.com/InyekakaB)*,*[***LinkedIn***](http://www.linkedin.com/in/christianinyekaka)*,*[***GitHub***](https://github.com/chrisBokotaII)**.**

[Smart Contracts](https://medium.com/tag/smart-contracts?source=post_page-----4a1be8795283---------------smart_contracts-----------------)

[Blockchain](https://medium.com/tag/blockchain?source=post_page-----4a1be8795283---------------blockchain-----------------)

[Ethereum Blockchain](https://medium.com/tag/ethereum-blockchain?source=post_page-----4a1be8795283---------------ethereum_blockchain-----------------)

[Nft](https://medium.com/tag/nft?source=post_page-----4a1be8795283---------------nft-----------------)

[Ethereum](https://medium.com/tag/ethereum?source=post_page-----4a1be8795283---------------ethereum-----------------)