1. Write a Solidity Smart Contract for a Non-Fungible  
   Token (NFT) called Virtual Land NFT which signifies a  
   Virtual Land in Metaverse.

Ans:- VirtuallLandNFT for Metaverse :

This is a Solidity Smart Contract for a (NFT) called VirtualLandNFT which signifies a virtual land in Metaverse .

It is based on the ERC721 standrad , for creating unique , indivisible assets .

The main points are :-

1. - *mintLand* - allows users to create a new virtual land token by specifying its name , price , and image URL .
2. –*setVirtualLandForSale ---* the owner of a token to put it sale .
3. purchaseVirtualLand- users to buy a virtual land token is currently for sale
4. The **getVirtualLandDetails**

Now this is a basic Solidity smart contract for a Non-Fungible Token (NFT) called Virtual Land NFT:

// Write a Solidity Smart Contract for a Non-Fungible Token (NFT) called Virtual Land NFT

//Which signifies a Virtual Land in Metaverse .

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "@openzeppelin/contracts/utils/Counters.sol";

contract VirtualLandNFT is ERC721 {

    using Counters for Counters.Counter;

    Counters.Counter private \_tokenIdCounter;

    uint public constant MAX\_SUPPLY = 500;

    struct VirtualLand {

        string name;

        address owner;

        uint price;

        string imageURL;

        bool forSale;

    }

    mapping(uint => VirtualLand) public virtualLands;

    constructor() ERC721("VirtualLandNFT", "VLNFT") {}

    function mintLand(string memory \_name, uint \_price, string memory \_imageURL) public {

        \_tokenIdCounter.increment();

        uint tokenId = \_tokenIdCounter.current();

        require(tokenId <= MAX\_SUPPLY, "The supply is limited");

        \_safeMint(msg.sender, tokenId);

        virtualLands[tokenId] = VirtualLand(\_name, msg.sender, \_price, \_imageURL, false);

    }

    function setVirtualLandForSale(uint tokenId) external {

        require(ownerOf(tokenId) == msg.sender, "Only owner can set for sale");

        virtualLands[tokenId].forSale = true;

    }

    function purchaseVirtualLand(uint tokenId) external payable {

        VirtualLand storage virtualLand = virtualLands[tokenId];

        require(virtualLand.forSale, "Land not for sale");

        require(msg.value >= virtualLand.price, "Insufficient balance");

        address previousOwner = virtualLand.owner;

        virtualLand.owner = msg.sender;

        virtualLand.forSale = false;

        payable(previousOwner).transfer(virtualLand.price);

        \_transfer(previousOwner, msg.sender, tokenId);

    }

    function getVirtualLandDetails(uint tokenId) public view returns (string memory, address, uint, string memory, bool) {

        VirtualLand memory land = virtualLands[tokenId];

        return (land.name, land.owner, land.price, land.imageURL, land.forSale);

    }

}