pragma solidity ^0.6.2; // SPDX-License-Identifier: MIT License import "./Context.sol"; contract Ownable is Context { address private \_owner; event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); /\*\* \* @dev Initializes the contract setting the deployer as the initial owner. \*/ constructor () public { address msgSender = \_msgSender(); \_owner = msgSender; emit OwnershipTransferred(address(0), msgSender); } /\*\* \* @dev Returns the address of the current owner. \*/ function owner() public view returns (address) { return \_owner; } /\*\* \* @dev Throws if called by any account other than the owner. \*/ modifier onlyOwner() { require(\_owner == \_msgSender(), "Ownable: caller is not the owner"); \_; } /\*\* \* @dev Leaves the contract without owner. It will not be possible to call \* `onlyOwner` functions anymore. Can only be called by the current owner. \* \* NOTE: Renouncing ownership will leave the contract without an owner, \* thereby removing any functionality that is only available to the owner. \*/ function renounceOwnership() public virtual onlyOwner { emit OwnershipTransferred(\_owner, address(0)); \_owner = address(0); } /\*\* \* @dev Transfers ownership of the contract to a new account (`newOwner`). \* Can only be called by the current owner. \*/ function transferOwnership(address newOwner) public virtual onlyOwner { require(newOwner != address(0), "Ownable: new owner is the zero address"); emit OwnershipTransferred(\_owner, newOwner); \_owner = newOwner; } }