

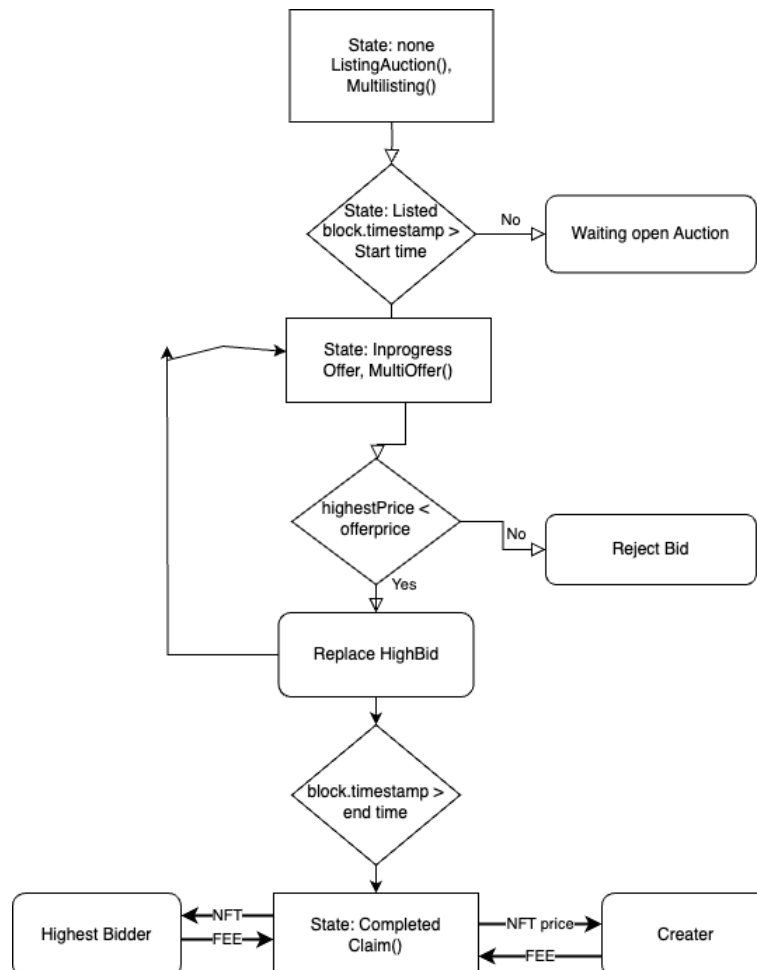
Up NFT Docs

Deployed Contract

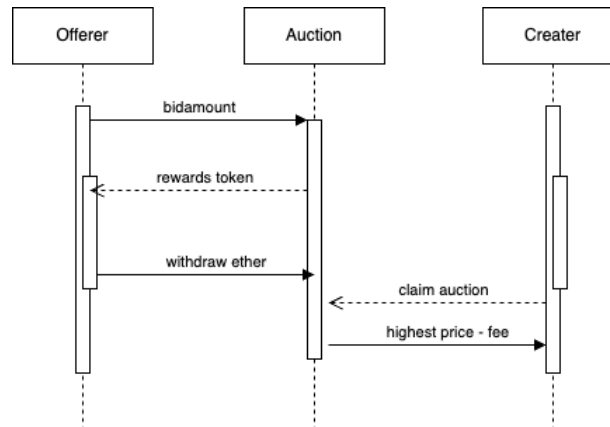
- AuctionProxy contract
 - [0x69831dfa35388291748ee4c63d80b7ebe40c4997](https://sepolia.basescan.org/address/0x69831dfa35388291748ee4c63d80b7ebe40c4997)
 - <https://sepolia.basescan.org/address/0x69831dfa35388291748ee4c63d80b7ebe40c4997>
- WDTOKEN contract
 - [0x75A8F001f36E0ad88A328C09cEd9274db8E89f72](https://sepolia.basescan.org/address/0x75a8f001f36e0ad88a328c09ced9274db8e89f72)
 - <https://sepolia.basescan.org/address/0x75a8f001f36e0ad88a328c09ced9274db8e89f72#code>
- Swap Token contract
 - [0x4b1c3407D22f89C512EA33e12C891b4B39384bdb](https://sepolia.basescan.org/address/0x4b1c3407d22f89c512ea33e12c891b4b39384bdb)
 - <https://sepolia.basescan.org/address/0x4b1c3407d22f89c512ea33e12c891b4b39384bdb#code>
- NFT Uptoken contract
 - [0x00b415708b13137ecF4F323F49A1079a0A85afeC](https://sepolia.basescan.org/address/0x00b415708b13137ecf4f323f49a1079a0a85afec)
 - <https://sepolia.basescan.org/address/0x00b415708b13137ecf4f323f49a1079a0a85afec#code>

프로젝트 상세

본 프로젝트는 사용자가 가지고 있는 NFT를 자유롭게 경매에 올려 입찰과 낙찰을 할 수 있는 컨트랙트입니다.



1. creator가 `auctionListing` 을 통해 NFT를 auction에 등록을 합니다.
이때 NFT가 auction에게 approve 된 상태여야 합니다.
auction에 있는 거래 수에 따라 `auctionId`가 결정되고 `state`가 `Listed`로 바뀝니다.
2. Listing된 auction이 start time에 도달하면 `state`가 `Inprogress`로 바뀝니다.
Offerer들이 bid가 가능해지며 다른 offerer가 최고 비드를 갱신하면 그전 최고 비드는 `bidReturn`에 쌓입니다.
3. 시간이 지나며 경매가 마감시간에 도달 하였을 경우 자동으로 종료 되며 `claim` 함수를 통해 NFT, 경매 금액을 받을 수 있다.
입찰자, 경매 생성자 둘 중 한명만 실행 시켜도 둘다에게 각각 자산이 이동된다.



경매 진행 과정에서 bid Offer가 생성되면 총 금액의 0.01 % 만큼 토큰을 제공한다.

그리고 경매가 마친후 nft 금액의 1%를 거래소 수수료로 받아 이더로 저장한다.

이 과정을 통해 토큰과 이더의 수량변화에 따라 변하는 swap protocol을 생성하여 두 개의 자산이 자유롭게 거래될 수 있도록 만든다.

함수 설명

core/AuctionProxy

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

import "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";

contract AuctionProxy is ERC1967Proxy {
    constructor(address contractLogic, bytes memory contractData) ERC1967Proxy(contractLogic, contractData) {}

    receive() external payable {}
}

```

openzeppelin의 ERC1967proxy library를 import 해와 proxy 컨트랙트를 구성했습니다.

- Input
 - address contractLogic, bytes memory contractData
 - 생성 인자로 logic 서버의 address와 같이 실행할 데이터를 넣습니다.
- 동작
 - 생성자로 넣은 주소를 impl slot에 저장하고 delegatecall로 data를 실행합니다.

core/AuctionLogic

```

/// proxy 서버의 owner를 딱 한번 초기화 시키는 함수
function initialize(address _swap) external initializer onlyProxy {
    auctionOwner = msg.sender;
    swap = SwapToken(payable(_swap));
    token = WDTOKEN(swap.token());
}

```

initializer library를 이용해서 Logic 서버에서 사용되는 값을 proxy에서 초기화 하기 위해 구현하였습니다.
 딱 한번 실행되며 proxy를 사용하는데 필요한 변수를 저장하는 용도입니다.

- Input
 - swapToken contract의 주소를 받아 참조를 합니다.
- 동작
 - msg.sender를 auctionowner로 받아 처음 배포해야할때 실행해야 하는 함수입니다.

```

/// bidReturn getter func
function getBidreturn() external view onlyProxy returns (uint256) {
    return (bidReturn[msg.sender]);
}

```

- output
 - msg.sender의 bidReturn mapping을 참조하여 받을 돈을 보여줍니다.
- 동작
 - msg.sender의 입찰 실패로 저장된 금액을 볼 수 있습니다.

```

function getTokenBalance() external view onlyProxy returns (uint256) {
    return (token.balanceOf(msg.sender));
}

```

- output
 - msg.sender의 WDToken의 잔액을 내보냅니다.
- 동작
 - msg.sender의 WDToken의 잔액을 볼 수 있습니다.

```

function getState(uint256 Id) external view onlyProxy returns (State) {
    return (auctionList[Id].status);
}

```

- input
 - 상태 조회를 원하는 auciton의 id를 입력받습니다.
- output
 - 상태 조회를 원하는 auciton의 status를 반환합니다.
- 동작
 - id를 받아 auctionList에서 mapping 한 후 status를 반환합니다.

```
function getauctionOwner() external view onlyProxy returns (address) {
    return (auctionOwner);
}
```

- output
 - 현재 auction contract의 owner를 반환합니다.

```
function getAuctionList(uint256 Id) external view onlyProxy returns (Auction memory) {
    require(Id < auctionCounter, "Id < auctionCounter");
    return auctionList[Id];
}
```

- input
 - 원하는 auction의 id를 입력받습니다.
- output
 - 원하는 auction의 구조체를 반환합니다.
- 동작
 - id를 받아 auctionList에서 Auction 구조체를 반환합니다. Id 가 실제 만들어진 counter보다 크면 안됩니다.

```
function nextState(uint256 Id) private whenNotPaused {
    auctionList[Id].status = State(uint8(auctionList[Id].status) + 1);
}
```

- input
 - 원하는 auction의 id를 입력받습니다.
- 동작
 - id를 받아 auctionList에서 State enum에 따라 한단계 나아갑니다.

```
function isERC721(address tokenAddress) private view returns (bool) {
    IERC165 tokens = IERC165(tokenAddress);

    try tokens.supportsInterface(0x80ac58cd) returns (bool isSupported) {
        return isSupported;
    } catch {
        return false;
    }
}
```

- input
 - 체크를 진행할 tokenAddress를 입력받습니다.
- output
 - token이 ERC721를 따르는지 체크 합니다.
- 동작
 - ERC721의 interface를 넣어 해당 contract가 따르는지 체크하고 결과를 반환합니다.

```
function listingAuction(address _tokenAddress, uint256 _tokenId, uint256 _startTime, uint256 _openPrice)
    external
    whenNotPaused
```

```

onlyProxy
returns (uint256 auctionId)
{
    require(_startTime > block.timestamp, "Start time is faster than block time");
    require(isERC721(_tokenAddress), "Only ERC721 Token");
    IERC721 nftToken = IERC721(_tokenAddress);
    require(nftToken.getApproved(_tokenId) == address(this), "NFT must approve to auction");
    if (nftToken.ownerOf(_tokenId) != msg.sender) {
        revert Errors.NotNFTOwner();
    }
    require(isListed[_tokenAddress][_tokenId] == false, "Already Listed NFT");
    auctionList[auctionCounter] = Auction({
        auctionId: auctionCounter,
        owner: payable(msg.sender),
        tokenAddress: _tokenAddress,
        tokenId: _tokenId,
        startTime: _startTime,
        openPrice: _openPrice,
        endTime: _startTime + ACUTION_TIME,
        highestBidder: address(0),
        highestPrice: _openPrice,
        status: State.Listed
    });
    isListed[_tokenAddress][_tokenId] = true; // list에 올라갔는지 체크
    auctionCounter += 1;
    return (auctionCounter - 1);
}

```

- input
 - 경매에 등록할 tokenaddress, tokenId, starttime, openprice를 받습니다
- output
 - 경매소에서 만들어진 경매의 Id를 반환합니다.
- 동작
 - 경매소 등록에 필요한 정보를 받아 등록합니다.
 - msg.sender는 경매 소유주와 일치해야 하고 경매 시작 시간은 현재 블록보다 나중이어야 합니다.
 - tokenAddress와 tokenId를 mapping 하여 중복 등록을 방지합니다.

```

function multiList(
    address[] memory tokenAddress,
    uint256[] memory tokenId,
    uint256[] memory startTime,
    uint256[] memory openPrice
) external whenNotPaused onlyProxy returns (uint256[] memory) {
    uint256 arrLen = tokenAddress.length;
    if ((arrLen != tokenId.length) || (arrLen != startTime.length) || (arrLen != openPrice.length)) {
        revert Errors.NotEqualEachArgument();
    }
    bytes[] memory callData = new bytes[](arrLen);
    for (uint256 i = 0; i < arrLen; i++) {
        callData[i] = abi.encodeWithSignature(
            "listingAuction(address,uint256,uint256,uint256)",
            tokenAddress[i],

```

```

        tokenId[i],
        startTime[i],
        openPrice[i]
    );
}
uint256[] memory tokenIds = new uint256[](arrLen);
bytes[] memory results;
(bool success, bytes memory data) =
    address(this).delegatecall(abi.encodeWithSignature("multicall(bytes[])", callData));
if (!success) revert Errors.FailDelegateCall();

results = abi.decode(data, (bytes[]));
for (uint256 i = 0; i < results.length; i++) {
    tokenIds[i] = abi.decode(results[i], (uint256));
}
return (tokenIds);
}

```

- input
 - 경매에 등록할 tokenaddress, tokenId, starttime, openprice를 배열로 받습니다
- output
 - 경매소에서 만들어진 경매의 Id 배열을 반환합니다.
- 동작
 - 여러 NFT를 한번에 경매 등록하는 함수입니다.
 - callData를 만들어 multicall로 보내 결과를 다시 decode하여 반환합니다.

```

function offerBid(uint256 Id) external payable whenNotPaused onlyProxy transitionState(Id) onlyInproress(I
d) {
    Auction memory info = auctionList[Id];
    require(info.highestPrice < msg.value, "Must Bid over the Highest Price");
    if (info.highestBidder != address(0)) {
        bidReturn[info.highestBidder] += info.highestPrice;
    }
    auctionList[Id].highestPrice = msg.value;
    auctionList[Id].highestBidder = msg.sender;
    // 입찰을 한다면 0.01 % token을 준다
    token.transfer(msg.sender, swap.getETHPriceInToken(msg.value / 10000));
}

```

- input
 - 경매에 offer할 auction id를 받습니다
- 동작
 - id를 통해 경매 정보를 받아 현재 가격과 입찰 가격을 비교합니다.
 - 현재 가격보다 크다면 최고가와 입찰자를 현재 상태로 바꿉니다.
 - offer만 해도 활성화를 위해 입찰가의 0.01% 를 토큰으로 바꿔 제공합니다

```

function offerBids(uint256 Id, uint256 amount)
    external
    payable
    whenNotPaused

```

```

onlyProxy
transitionState(Id)
onlyInproress(Id)
{
    Auction memory info = auctionList[Id];
    require(info.highestPrice < amount, "Must Bid over the Highest Price");
    require(amount < msg.value, "Msg value is low than amount");
    if (info.highestBidder != address(0)) {
        bidReturn[info.highestBidder] += info.highestPrice;
    }
    auctionList[Id].highestPrice = amount;
    auctionList[Id].highestBidder = msg.sender;
    // 입찰을 한다면 0.01 % token을 준다
    token.transfer(msg.sender, swap.getETHPriceInToken(amount / 10000));
}

```

- input
 - 경매에 offer할 auction id와 입찰 금액을 받습니다
- 동작
 - 입찰 금액을 msg.value가 아니고 amount를 통해 받습니다. 주로 multioffer에서 사용합니다
 - id를 통해 경매 정보를 받아 현재 가격과 입찰 가격을 비교합니다.
 - 현재 가격보다 크다면 최고가와 입찰자를 현재 상태로 바꿉니다.
 - offer만 해도 활성화를 위해 입찰가의 0.01% 를 토큰으로 바꿔 제공합니다

```

function multiOffer(uint256[] memory ids, uint256[] memory bids)
    external
    payable
    onlyProxy
    whenNotPaused
    returns (bool)
{
    if (ids.length != bids.length) revert Errors.NotEqualEachArgument();
    uint256 bidsamount;
    uint256 idlength = ids.length;
    bytes[] memory callData = new bytes[](idlength);
    for (uint256 i = 0; i < idlength; i++) {
        callData[i] = abi.encodeWithSignature("offerBids(uint256,uint256)", ids[i], bids[i]);
        bidsamount += bids[i];
    }
    require(bidsamount == msg.value, "Not same bids, value");

    (bool success,) = address(this).delegatecall(abi.encodeWithSignature("multicall(bytes[])", callData));

    if (!success) revert Errors.FailDelegateCall();
    return (success);
}

```

- input
 - offer할 auction id와 bids 배열을 받습니다
- output
 - 성공 여부를 반환합니다

- 동작
 - 다양한 오퍼를 한번에 실행 합니다.
 - multicall 방식으로 offerBids 함수를 실행합니다.

```
function cancelAuction(uint256 Id) external onlyProxy transitionState(Id) onlyNftOwner(Id) {
    Auction memory info = auctionList[Id];
    require(info.status == State.Listed, "Only Cancel Listed state");
    auctionList[Id].status = State.Canceled;
}
```

- input
 - 취소할 auction id를 받습니다
- 동작
 - 취소를 하기 위해서는 auction의 상태가 listed여야 합니다. 진행중이거나 완료된 경매는 취소불가능 합니다.
 - 경매의 소유자만 이 함수를 실행할 수 있습니다.
 - 경매의 상태를 canceled로 바꿉니다.

```
function withdrawBid() external onlyProxy {
    uint256 amount = bidReturn[msg.sender];
    if (amount > 0) {
        bidReturn[msg.sender] = 0;
        bool success = payable(msg.sender).send(amount);
        if (!success) {
            revert Errors.transferError();
        }
    }
}
```

- 동작
 - 저장된 bidReturn 값을 다시 보내줍니다.

```
function claim(uint256 Id) external onlyProxy nonReentrant transitionState(Id) onlyCompleted(Id) returns (bool) {
    Auction memory info = auctionList[Id];
    IERC721 nftToken = IERC721(info.tokenAddress);

    require(msg.sender == info.highestBidder, "Must HighestBidder");
    require(nftToken.getApproved(info.tokenId) == address(this), "NFT must approve to auction"); // 토큰의 i
    d가 Auction contract에게 approve 되어 있어야 함

    if (info.owner == info.highestBidder) return (false); // owner = bidder 같을 경우 false
    nftToken.safeTransferFrom(info.owner, info.highestBidder, info.tokenId); // 토큰 bidder가 ca일 경우 onerc7
    21Received 함수 있어야 함 없으면 터짐

    /// auction이 수수료 0.1% 가져감... ㅎㅎ 이돈은 좋은곳에 쓰일예정
    uint256 fee = info.highestPrice / 1000;
    (bool success,) = info.owner.call{value: info.highestPrice - fee}("");
    if (!success) {
        revert Errors.transferError();
    }
    swap.swapETHToken{value: fee}();
}
```



```
// 입찰 완료 되면 총 금액의
return (true);
}
```

- input
 - claim할 auction id를 입력 받습니다.
- output
 - 성공 여부를 반환합니다
- 동작
 - bidder와 owner 중 한명만 실행해도 거래가 진행됩니다.
 - 최고가 bidder에게 nft를 보내고, owner에게 bidprice를 보냅니다.
 - bidprice의 1%를 수수료로 받습니다.

```
function emergencyWithdraw(uint256 Id) external onlyProxy whenPaused {
    Auction memory info = auctionList[Id];

    require(msg.sender == info.highestBidder, "Must HighestBidder");
    if (info.status == State.Inprogress) {
        // 진행중인 경매일 때 최고 비드 에게 돈 환불
        auctionList[Id].status = State.Canceled;
        bool success = payable(info.highestBidder).send(info.highestPrice);
        if (!success) {
            revert Errors.transferError();
        }
    }
}
```

- input
 - 작동할 auction id를 입력받습니다.
- 동작
 - Auction contract가 pause한 경우에만 작동하며 Highest Bidder만 실행 가능합니다.
 - 경매 중 pause가 발생하면 입찰 금액을 환불해줍니다.

```
function stopContract() external override onlyProxy onlyAuctionOwner {
    _paused = true;
    emit Paused(msg.sender);
}
```

- 동작
 - Auction Owner가 긴급한 상황에 contract를 멈출수있는 함수입니다.
 - pause가 되면 출금 기능만 가능합니다.

```
function resumeContract() external override onlyProxy onlyAuctionOwner whenPaused {
    _paused = false;
    emit Unpaused(msg.sender);
}
```

- 동작

- Auction Owner가 contract 다시 Resume 할 수 있는 함수입니다.

```
function multicall(bytes[] calldata data) external payable returns (bytes[] memory results) {
    bool success;

    results = new bytes[](data.length);
    for (uint256 i = 0; i < data.length; i++) {
        (success, results[i]) = address(this).delegatecall(data[i]);
        if (!success) revert Errors.FailDelegateCall();
    }
    return results;
}
```

- input
 - Multicall에서 사용될 인코딩된 function selector와 인자를 받습니다.
- output
 - Multicall에 반환된 결과 값을 반환합니다.
- 동작
 - delegatecall로 멀티콜을 실행합니다.
 - 실패한다면 Error를 반환합니다.

util/AuctionNFT

```
contract UpToken is ERC721, Multicall, Pausable {
    uint256 private _tokenIdCounter;
    WDTOKEN private _token;
    address _auction;

    constructor(address token, address auction) ERC721("UpNFT", "UT") {
        _tokenIdCounter = 1;
        _token = WDTOKEN(token);
        _auction = auction;
    }
}
```

- input
 - UpToken을 민팅할 때 사용할 WDTOKEN과 거래소인 auction 주소를 받습니다.
- 동작
 - NFT를 만들 수 있는 컨트랙트를 만듭니다.
 - 이를 통해 WDTOKEN의 보상으로 NFT를 민팅할 수 있게 만듭니다.

```
function mint(address to) public returns (uint256) {
    uint256 tokenId = _tokenIdCounter;
    uint256 amount = _token.allowance(msg.sender, address(this));
    require(amount > 0.0001 ether, "More WD token need!");
    _token.transferFrom(msg.sender, _auction, 0.0001 ether);
    _safeMint(to, tokenId);

    _tokenIdCounter++;
    return (tokenId);
}
```

- input
 - NFT를 받을 수 있는 주소를 받습니다.
- output
 - 만들어진 tokenId를 반환합니다
- 동작
 - NFT 민팅을 진행하는데 msg.sender가 미리 이 컨트랙트에게 approve를 정해진 토큰 수만큼 해야 합니다.
 - 이를 통해 이 contract는 auction에게 토큰을 전송합니다.

```
function multiMint(address[] memory dst) public returns (uint256[] memory) {
    uint256 dstLen = dst.length;
    bytes[] memory results;
    bytes[] memory callData = new bytes[](dstLen);
    uint256[] memory tokenIds = new uint256[](dstLen);
    for (uint256 i = 0; i < dstLen; i++) {
        callData[i] = abi.encodeWithSignature("mint(address)", dst[i]);
    }
    (bool success, bytes memory data) =
        address(this).delegatecall(abi.encodeWithSignature("multicall(bytes[])", callData));
    if (!success) revert();

    results = abi.decode(data, (bytes[]));
    for (uint256 i = 0; i < results.length; i++) {
        tokenIds[i] = abi.decode(results[i], (uint256));
    }
    return (tokenIds);
}
```

- input
 - NFT를 받을 수 있는 주소 배열을 받습니다.
- output
 - 만들어진 tokenId 배열을 반환합니다
- 동작
 - 여러 NFT를 만들기 위해 multicall로 mint를 진행합니다.

util/AuctionToken

```
contract WDTOKEN is ERC20 {
    constructor() ERC20("WDETH", "WD") {
        _mint(msg.sender, 10 ** decimals() * 1); // auction 에게 1 * 10**18 개 발행
    }
}
```

- 거래소에서 보상과 NFT 민팅에 사용될 WDTOKEN을 만듭니다.
- 1 ether 개수만큼 만듭니다.

```
contract SwapToken is Pausable {
    WDTOKEN public token;

    constructor(address _token) {
```

```

    token = WDTOKEN(_token);
}

```

- Swap 해주는 contract로 이더와 token의 비율에 따라 조절됩니다.

```

function swapETHToken() external payable {
    uint256 ethReserve = address(this).balance;
    uint256 tokenReserve = token.balanceOf(address(this));

    uint256 ethIn = msg.value;
    uint256 tokenOut = (ethIn * tokenReserve) / (ethReserve + ethIn);

    require(tokenReserve >= tokenOut, "Not enough tokens");

    ethReserve += ethIn;
    tokenReserve -= tokenOut;

    token.transfer(msg.sender, tokenOut);
}

```

- 동작
 - Eth를 받아 Token을 주는 함수입니다.
 - 이더 수와 token의 공급이 일정한 값이 되도록 만든 AMM입니다.

```

function swapTokenETH(uint256 amount) external {
    uint256 ethReserve = address(this).balance;
    uint256 tokenReserve = token.balanceOf(address(this));

    uint256 tokenIn = amount;
    require(tokenIn <= token.allowance(msg.sender, address(this)), "Must approve amount");
    token.transferFrom(msg.sender, address(this), tokenIn);
    uint256 ethOut = (tokenIn * ethReserve) / (tokenReserve + tokenIn);

    require(ethReserve >= ethOut, "Not enough ethers Sorry");

    ethReserve -= ethOut;
    tokenReserve += tokenIn;

    bool success = payable(msg.sender).send(ethOut);
    require(success, "Something wrong through sending");
}

```

- input
 - amount만큼 token eth로 교환하는 양을 받는다.
- 동작
 - 토큰을 받아 approve 되어있는지 확인후 transferfrom으로 토큰을 받는다.
 - 그리고 이더로 바꾸어 내보낸다.

```

function getTokenPriceInETH(uint256 tokenAmount) external view returns (uint256) {
    uint256 ethReserve = address(this).balance;
    uint256 tokenReserve = token.balanceOf(address(this));
}

```

```
uint256 ethOut = (tokenAmount * ethReserve) / (tokenReserve + tokenAmount);
return ethOut;
}
```

- input
 - token을 환전하고 싶은 만큼 입력한다.
- output
 - 토큰의 수에 따라 이더의 수를 반환한다.
- 동작
 - AMM에 따라 이더의 수를 계산하여 반환한다.

```
function getETHPriceInToken(uint256 etherAmount) external view returns (uint256) {
    uint256 ethReserve = address(this).balance;
    uint256 tokenReserve = token.balanceOf(address(this));

    uint256 tokenAmount = (etherAmount * tokenReserve) / (ethReserve + etherAmount);
    return tokenAmount;
}
```

- input
 - 이더를 환전하고 싶은 만큼 입력한다.
- output
 - 이더의 수에 따라 환전되는 토큰의 수를 반환한다.
- 동작
 - AMM에 따라 토큰의 수를 계산하여 반환한다.

util/Pausable

```
function paused() public view virtual returns (bool) {
    return _paused;
}
```

- output
 - 현재 _paused의 값을 반환한다.
- 동작
 - _paused 변수에 접근하여 값을 반환한다.

```
function _requireNotPaused() internal view virtual {
    if (paused()) {
        revert EnforcedPause();
    }
}
```

- 동작
 - contract가 멈춰있다면 에러를 표출한다.

```
function _requirePaused() internal view virtual {
    if (!paused()) {
        revert ExpectedPause();
    }
}
```

```
}  
}
```

- 동작
 - contract가 멈춰있지 않는다면 에러를 표출한다.

```
function stopContract() external virtual onlyOwner whenNotPaused {  
    _paused = true;  
    emit Paused(msg.sender);  
}
```

- 동작
 - contract를 멈춰 _pause 변수 값을 true로 바꾸고 이벤트를 표출한다.

```
function resumeContract() external virtual onlyOwner whenPaused {  
    _paused = false;  
    emit Unpaused(msg.sender);  
}
```

- 동작
 - contract를 다시 시작하여 _pause 변수 값을 바꾸고 이벤트를 표출한다.

```
function emergencyTransfer(uint256 amount) external whenPaused onlyOwner {  
    require(amount <= address(this).balance, "Too much ether");  
    bool success = payable(msg.sender).send(amount);  
    require(success, "send Error");  
}
```

- input
 - 긴급 인출할 금액을 입력받는다.
- 동작
 - 컨트랙트가 멈춰있을 경우 인출할 금액을 받아 보내준다.