# Catgirl Marketplace

## Project Overview

Catgirl Marketplace is an NFT marketplace based on the Wyvern Protocol contract where users are able to buy and sell their NFT. Catgirl contracts allow users to list, buy, sell, and cancel orders.

# 1. Functional Requirement

## 1.1 Roles

- **UPGRADER_ROLE:** Catgirl contracts are UUPSUpgradeable contracts, so dev has a role to upgrade the contract.
- **SETTER_ROLE:** able to change some settings of contracts.
- **User**: who can buy and sell NFT through contracts.

## 1.2 Features

Catgirl Marketplace has the following features:

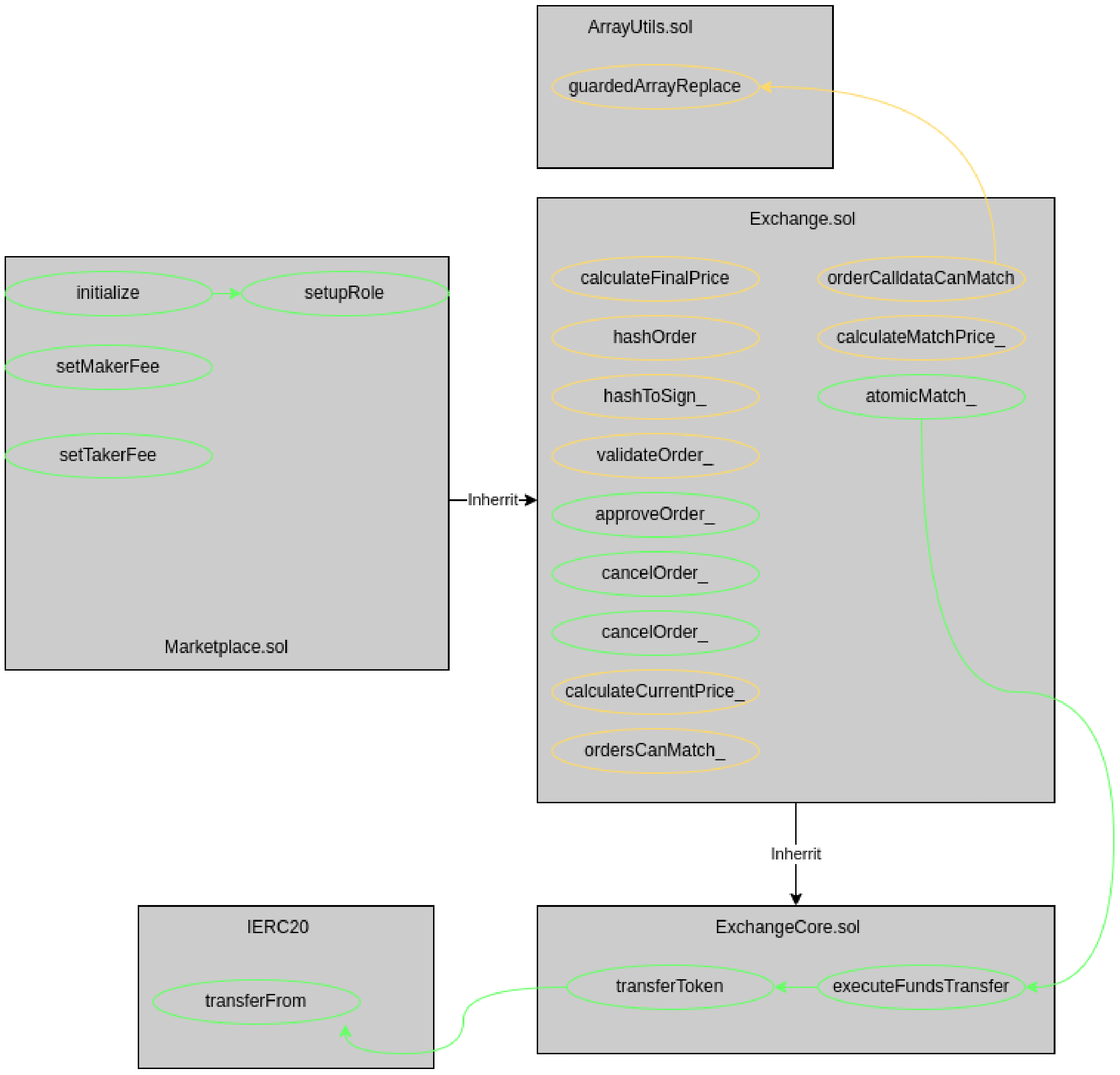
- Listing a single/bundle of NFT with a fixed price
- Listing a single/bundle of NFT as an auction
- Cancellation listing.
- Cancellation bidding.
- Upgrade contract to the new version (Upgrader role)
- Setting new fee protocol (Setter role)

## 1.3 Use cases

- Listing a single/bundle of NFT as a fixed price method (BNB or any ERC20 token), the seller determines the price, if any other user wants to buy those with that price, they could pay the price and get the NFT.
- Listing single/bundle of NFT as auction method (BNB or any ERC20 token). A buyer would need to make the bid to the NFT, the seller could accept any bid from buyers. If the auction end and the seller hasn't accepted any bid, the highest bid over the reserve price that the seller proposed would be accepted.
- Cancellation listing: After listing NFT, the seller can cancel the listing, and the buyer can not buy the NFT anymore.
- Cancellation bid: After bidding NFT in an auction, the buyer can cancel their bid, and the seller can not accept the cancelled bid.

# 2. Technical Requirement

## 2.1 Architecture Overview



## 2.2 Contract information

### 2.2.1 ExchangeCore.sol

#### 2.2.1.1 Assets

ExchangeCore contract contains 2 structs:

- **Sig**: This object contains information about the signature of the user

    - **uint8 v:** v parameter in a signature structure.
    - **bytes32 r:** r parameter in a signature structure.
    - **bytes 32 s:** s parameter in a signature structure.
- **Order:** An order on the exchange
    - **address exchange**: address of exchange contract
    - **address maker**: Order maker address
    - **address take:** Order taker address if specified.
    - **uint makerRelayerFee**: Maker relayer fee of the order, unused for taker order.
    - **uint takerRelayerFee**: Taker relayer fee of the order, or maximum taker fee for a taker order.
    - **address feeRecipient**: Order fee recipient or zero address for taker order.
    - **enum side**: Side (buy/sell).
    - **enum saleKind**: Kind of sale (fixed price/auction).
    - **enum howToCall**: Call/DeletageCall
    - **address target**: target
    - **bytes callData**: call data
    - **bytes replacementPattern**: Calldata replacement pattern, or an empty byte array for no replacement.
    - **address paymentToken**: Token used to pay for the order, or the zero-address as a sentinel value for Ether.
    - **uint basePrice**: Base price of the order (in paymentTokens)
    - **uint listingTime**: Listing timestamp
    - **uint expirationTime**: Expiration timestamp - 0 for no expiry.
    - **uint salt**: Order salt, used to prevent duplicate hashes.

Besides the mentioned structs, the following entities are present in the contract:

- **INVERSE_BASIS_POINT:** Constant denominator for %100.
- **cancelledOrFinalized:** a public mapping of cancelled/finalised orders, by hash.
- **approvedOrders:** a public mapping of orders verified by on-chain approval (alternative to ECDSA signatures so that smart contracts can place orders directly)

#### 2.2.1.2 Modifier

ExchangeCore contract does not have any modifier.

### 2.2.1.3 Functions

- **transferTokens(address token, address from, address to, uint amount)**: Transfer tokens
- **sizeOf(Order memory order)**: Calculate size of an order struct when tightly packed
- **hashOrder(Order memory order)**: Hash an order, returning the canonical order hash, without the message prefix
- **hashToSign(Order memory order)**: Hash an order, returning the hash that a client must sign, including the standard message prefix
- **requireValidOrder(Order memory order, Sig memory sig)**: Assert an order is valid and return its hash
- **validateOrderParameters(Order memory order)**: Validate order parameters (does not check signature validity)
- **validateOrder(bytes32 hash, Order memory order, Sig memory sig)**: Validate a provided previously approved / signed order, hash, and signature.
- **approveOrder(Order memory order, bool orderbookInclusionDesired)**: Approve an order and optionally mark it for orderbook inclusion. Must be called by the maker of the order
- **cancelOrder(Order memory order, Sig memory sig)**: Cancel an order, preventing it from being matched. Must be called by the maker of the order
- **calculateCurrentPrice(Order memory order)**: Calculate the current price of an order (convenience function)
- **calculateMatchPrice(Order memory buy, Order memory sell)**: Calculate the price two orders would match at, if in fact they would match (otherwise fail)
- **executeFundsTransfer(Order memory buy, Order memory sell)**: Execute all ERC20 token / Ether transfers associated with an order match (fees and buyer => seller transfer)
- **ordersCanMatch(Order memory buy, Order memory sell):** Return whether or not two orders can be matched with each other by basic parameters (does not check order signatures / calldata or perform static calls)
- **execute(address to, uint256 value, bytes memory data, HowToCall operation):** call to target.
- **atomicMatch(Order memory buy, Sig memory buySig, Order memory sell, Sig memory sellSig)**: Atomically match two orders, ensuring validity of the match, and execute all associated state transitions. Protected against reentrancy by a contract-global lock.

### 2.2.2 Exchange.sol

This contract inherits from the ExchangeCore contract and implements some other functions.

#### 2.2.2.1 Assets

- **uint MAKER_RELAYER_FEE**: value of fee from maker when order matched.
- **uint MAKER_RELAYER_FEE**: value of fee from maker when order matched.

#### 2.2.2.2 Functions

- **calculateFinalPrice(SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind, uint basePrice, uint extra, uint listingTime, uint expirationTime)**:Call calculateFinalPrice - library function exposed for testing.
- **hashOrder_(address[6] memory addrs, uint[4] memory uints,SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind, HowToCall howToCall, bytes memory callData, bytes memory replacementPattern)**: Call hashOrder - Solidity ABI encoding limitation workaround.
- **hashToSign_(address[6] memory addrs, uint[4] memory uints, SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind,HowToCall howToCall, bytes memory callData, bytes memory replacementPattern)**: Call hashToSign from ExchangeCore
- **validateOrder_( address[6] memory addrs, uint[4] memory uints, SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind, bytes memory callData, bytes memory replacementPattern, uint8 v, bytes32 r, bytes32 s)**: call validateOrder
- **approveOrder_(address[6] memory addrs, uint[4] memory uints, SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind, HowToCall howToCall, bytes memory callData, bytes memory replacementPattern, bool orderbookInclusionDesired)**: Call approveOrder
- **cancelOrder_(address[6] memory addrs, uint[4] memory uints, SaleKindInterface.Side side, SaleKindInterface.SaleKind saleKind, HowToCall howToCall, bytes memory callData, bytes memory replacementPattern, uint8 v, bytes32 r, bytes32 s):** Call cancelOrder
- **calculateCurrentPrice_(address[6] memory addrs, uint[4] memory uints, SaleKindInterface.Side side, SaleKindInterface.SaleKind**

**saleKind, HowToCall howToCall, bytes memory callData, bytes memory replacementPattern):** Call calculateCurrentPrice

- **ordersCanMatch_(address[12] memory addrs, uint[8] memory uints, uint8[6] memory saleKinds, bytes memory calldataBuy, bytes memory calldataSell, bytes memory replacementPatternBuy, bytes memory replacementPatternSell)**: Call ordersCanMatch
- **orderCalldataCanMatch(bytes memory buyCalldata, bytes memory buyReplacementPattern, bytes memory sellCalldata, bytes memory sellReplacementPattern)**: Return whether or not two orders' calldata specifications can match.
- **calculateMatchPrice_(address[12] memory addrs, uint[8] memory uints, uint8[6] memory saleKinds, bytes memory calldataBuy, bytes memory calldataSell, bytes memory replacementPatternBuy, bytes memory replacementPatternSell)**: call calculateMatchPrice
- **atomicMatch_(address[12] memory addrs, uint[8] memory uints, uint8[6] memory saleKinds, bytes memory calldataBuy, bytes memory calldataSell, bytes memory replacementPatternBuy, bytes memory replacementPatternSell, uint8[2] memory vs, bytes32[4] memory rssMetadata)**: Call automicMatch function.

### 2.2.3 Marketplace.sol

This contract inherits Exchange.sol, AccessControlUpgradeable, UUPSUpgradeable.

#### 2.2.3.1 Assets

- **string public constant name:** name of contract.
- **string public constant version**: version of contract, starting from 1.0
- **bytes32 public constant UPGRADER_ROLE:** hashed string name for Upgrade role.
- **bytes32 public constant SETTER_ROLE:** hashed string name for Setter role.

#### 2.2.3.1 Functions

- **initialize()**: initialised function of contract.
- **_authorizeUpgrade(address newImplementation)**: Update new implementation when contract needs to upgrade.
- **setMakerFee(uint256 makerFee)**: setting new fee for maker.
- **setTakerFee(uint256 takerFee)**: setting new fee for taker.

### 2.2.4 ArrayUtils.sol

This library contains various helpful functions to manipulate with Array.

#### 2.2.4.1 Functions

- **guardedArrayReplace(bytes memory array, bytes memory desired, bytes memory mask)**: Replace bytes in an array with bytes in another array, guarded by a bitmask.
- **arrayEq(bytes memory a, bytes memory b)**: Test if two arrays are equal
- **arrayDrop(bytes memory _bytes, uint _start)**: Drop the beginning of an array
- **arrayTake(bytes memory _bytes, uint _length)**: Take from the beginning of an array
- **arraySlice(bytes memory _bytes, uint _start, uint _length)**: Slice an array
- **unsafeWriteBytes(uint index, bytes memory source)**: Unsafe write byte array into a memory location
- **unsafeWriteAddress(uint index, address source)**: Unsafe write address into a memory location
- **unsafeWriteUint(uint index, uint source)**: Unsafe write uint into a memory location
- **unsafeWriteUint8(uint index, uint8 source)**: Unsafe write uint8 into a memory location

### 2.2.5 SaleKindInterface.sol

This library defined some enum and some validation functions would be used in Exchange contract.

#### 2.2.5.1 Assets

- **enum Side { Buy, Sell }**: 2 types of side, user who buys NFT and user who sells NFT.
- **enum SaleKind { FixedPrice, DutchAuction }**: 2 types of a sale, fixed price and auction.

#### 2.2.5.2 Functions

- **validateParameters(SaleKind saleKind, uint expirationTime)**: Check whether the parameters of a sale are valid
- **canSettleOrder(uint listingTime, uint expirationTime)**: Return whether or not an order can be settled

- **calculateFinalPrice(Side side, SaleKind saleKind, uint basePrice, uint extra, uint listingTime, uint expirationTime)**: Calculate the settlement price of an order

### 2.2.6 ReentrancyGuarded.sol

This contract has 1 modifier to prevent Reentracy attack.

#### 2.2.6.1 Modifiers

- **reentrancyGuard:** Prevent a contract function from being reentrant-called.