15 May 2024

Color v1.0

Prepared by Alexander Campbell

Project Access

Github: https://github.com/alexcampbelling/ColorMarketplace

On chain development deployment:
https://sepolia.etherscan.io/address/0xedf835f363617a9a8e5ecef4ca5887411b5db4d8#code

User flow

Direct vs Auction Listing

Direct Listing
In a direct listing, the seller sets a fixed price for the item. Buyers can purchase the item immediately at this price. Once a buyer agrees to the set price and completes the transaction, the sale is final.

However a user may place a bid on the direct listing for *under* the asking price. The lister may take this deal if they so desire.

You can send *buy* or *offer* on a direct listing. If an offer is made, the direct listing lister must use *acceptOffer*.

Auction Listing:
In an auction listing, the seller sets a starting price, and potential buyers place bids to purchase the item. The item is sold to the highest bidder at the end of the auction period. In some cases, the seller may also set a reserve price, which is the minimum amount they are willing to accept for the item. If the bidding does not reach the reserve price, the item is not sold. Additionally, the seller may also set a buyout price, which allows a buyer to bypass the auction process and purchase the item immediately at that price.

You can only make bids via the *offer* function, however if you hit the *buyoutPricePerToken* then this acts as an instant sell. The winning bidder and the auction lister must run *closeAuction* both to claim their portion of the transaction.

Escrow

In an auction, the NFT to be sold is placed in escrow at the start of the auction. This escrow location is simply the market contract, ensuring that the seller cannot back out or sell the NFT elsewhere once the auction has started.

If an offer is made to a direct listing, these funds are not held in escrow.

Admin endpoints

setPlatformFeeInfo
This function sets the platform fee recipient and the platform fee in basis points. The platform fee recipient is the address that will receive the platform fees, and the platform fee in basis points is the percentage of the transaction that will be taken as a fee.

setAuctionBuffers
This function sets the auction time buffer and the bid buffer in basis points. The auction time buffer is the extra time added to the auction when a bid is made near the end of the auction, and the bid buffer is the minimum percentage increase for a new bid to be valid.

erc20WhiteListAdd
This function adds a token to the ERC20 whitelist, allowing it to be used in the marketplace. The token address is the address of the ERC20 token to be added to the whitelist.

erc20WhiteListRemove
This function removes a token from the ERC20 whitelist, preventing it from being used in the marketplace. The token address is the address of the ERC20 token to be removed from the whitelist.

Expected use flow examples

List direct - buy at listed price
Seller: createListing() - Lists the item at a fixed price.
Buyer: buy() - Buys the item at the listed price.

List direct - bid under - accept bid
Seller: createListing() - Lists the item at a fixed price.
Buyer: offer() - Makes an offer under the listed price.
Seller: acceptOffer() - Accepts the offer made by the buyer.

List auction - buy at buyout price
Seller: createListing() - Lists the item for auction.
Buyer: offer() - Buys the item at the buyout price, bypassing the auction.

List auction - bid - auction ends
Seller: createListing() - Lists the item for auction.
Buyer: offer() - Places a bid on the item.
The set end time expires.
Buyer: closeAuction() - Claims token.
Seller: closeAuction() - Claims payment.

List auction - bid - outbid - auction ends
Seller: createListing() - Lists the item for auction.
Buyer 1: offer() - Places a bid on the item.
Buyer 2: offer(item, higherBidPrice) - Places a higher bid on the item, outbidding Buyer 1. Buyer 1 gets escrowed funds returned.
The set end time expires.
Buyer 2: closeAuction() - Claims token.
Seller: closeAuction() - Claims payment.

Story Protocol

- Royalties module is not the same as NFT royalties from ERC2981. They are set up as vaults for pooling money which goes to set people for emolument.
- License tokens are potentially non transferable if set in their token metadata. This halts them from calling their transfer functions.
    - Technically we can check for this, but there was not enough time contracted to fully implement this feature. Skeleton code exists in the *validateStoryProtocolCompliance()* function.
    - The effects of this are:
        - Auctions are not able to be created as they must be transferred to create the listing.

        - Direct listings potentially can be created, but they can not be transferred in the event someone attempts to buy them.
    - This implementation is likely to change with Story moving to an L1 as I can see it already has recently in the core v1 repo. So not having this working now is not a large issue as the Story team ramps up their development.
- It is possible to pause a token to stop royalty payments be claimed, and this was said to be discussed by the Story team, so no work was implemented for this yet.

Technically Color v1.0 is fully story compliant, however I feel some things may have to be updated as Story protocol continues development.

Events

ReceivedEther: Emitted when the contract receives Ether. Useful for tracking and auditing.

ListingAdded: Emitted when a new listing is added to the marketplace. Helps track new listings and notify interested parties.

ListingUpdated: Emitted when a listing is updated. Helps track changes to listings and notify interested parties.

ListingRemoved: Emitted when a listing is removed from the marketplace. Helps track when listings are removed and notify interested parties.

NewSale: Emitted when a sale occurs. Helps track sales, calculate fees, and notify interested parties.

NewOffer: Emitted when a new offer is made on a listing. Helps track offers and notify the seller or other interested parties.

AuctionClosed: Emitted when an auction is closed. Helps track the end of auctions, determine the winning bidder, and notify interested parties.

PlatformFeeInfoUpdated: Emitted when the platform fee information is updated. Helps track changes to the platform fee and notify interested parties.

AuctionBuffersUpdated: Emitted when the auction buffers (time buffer and bid buffer) are updated. Helps track changes to the auction parameters and notify interested parties.

Error messages

NotListingOwner(): The caller is not the owner of the listing.
DoesNotExist(): The item does not exist.

TokenNotSupported(): The token is not supported.
InvalidQuantity(): The quantity specified is invalid.
InvalidStartTime(): The start time specified is invalid.
TokenNotValidOrApproved(): The token is not valid or approved.
InvalidPrice(): The price specified is invalid.
ListingAlreadyStarted(): The listing has already started.
NotDirectListing(): The listing is not a direct listing.
InvalidTotalPrice(): The total price specified is invalid.
InvalidTokenAmount(): The token amount specified is invalid.
NotWithinSaleWindow(): The operation is not within the sale window.
InvalidMsgValue(): The message value is invalid.
InsufficientBalanceOrAllowance(): The balance or allowance is insufficient.
OfferExpired(): The offer has expired.
InactiveListing(): The listing is not active.
InvalidCurrency(): The currency specified is invalid.
ZeroAmountBid(): The bid amount is zero.
ValueNotNeeded(): The value is not needed.
NotWinningBid(): The bid is not the winning bid.
AuctionNotEnded(): The auction has not ended.
InvalidPlatformFeeBps(): The platform fee in basis points is invalid.
InvalidBPS(): The basis points specified are invalid.
InsufficientTokensInListing(): There are insufficient tokens in the listing.
NoTokensInListing(): There are no tokens in the listing.
NotAnAuction(): The listing is not an auction.
NotListingCreator(): The caller is not the creator of the listing.
InvalidERC20(): The ERC20 token is invalid.
TokenNotAccepted(): The token is not accepted.
InputLengthMismatch(): The input lengths do not match.
NotStoryCompliant(): The operation is not compliant with the story.

Deployment

1. Fill in environment variables by first running `cp .env.example .env` then filling in the `.env` file.
2. Locate the deployment script and update your requirement constructor arguments. There is a description for each with an example already written.
3. Run `forge clean` to ensure no extra contracts will be included in the development. This can sometimes happen after running tests.
4. Run `forge build` to compile the solidity into byte code.
5. Run `forge script ColorMarketPlaceDeploy --rpc-url $SEPOLIA_RPC_URL --verify --etherscan-api-key $ETHERSCAN_API_KEY --broadcast` to deploy Color to Sepolia.
    - This will deploy from your .env private key address. Details of where will be printed. Verification on etherscan is automatically run after deployment.

15 May 2024

Possible Extensions

- Upgradable contracts (https://docs.openzeppelin.com/upgrades-plugins/1.x/)
- Gas usage improvements
- Error messages having more details (useful for metrics)

Design decisions

- Cancel your own auction mid way through
    - Possibly frustrates people who have bid
    - Allows lister to retrieve their escrowed nft if not being bid on enough
- More admin control
    - Admin can release all escrowed items / funds in the market?
    - Block list for listers / nfts