NFTMarketplace Solidity Contract

by M for GenesisL1

Contents

1 Introduction

This document provides an in-depth discussion of the `NFTMarketplace.sol` smart contract. The contract is designed to function as a generic ERC721 NFT marketplace with specialized features, including:

- An **Editor Role** with administrative capabilities.
- A **Vault** mechanism, whereby any NFT transferred directly to the contract is stored under marketplace control.
- Support for **whitelisted** NFT contracts only.
- A **profit collector** address that receives proceeds from sales and fees.
- Two selling modes:
  - **Fixed-price listings** with a configurable listing fee.

  - **Auctions** with an adjustable fee, minimal bid increments, and a maximum number of bids.

- **Batch transferring** of vaulted NFTs to external addresses.
- Editor-driven bulk actions (e.g., mass listing, mass auctioning) for NFTs in the vault.

Below is the full source code, followed by a detailed explanation of its components.

2 Source Code

https://github.com/alpha-omega-labs/NFTMarketplace/blob/main/marketplace.sol

3 Detailed Explanation

This section describes each feature of the contract in detail.

3.1 Editor Role

- The contract maintains a private mapping `_editors` of addresses to booleans, indicating whether they hold the editor role.
- The constructor grants the deployer an **editor** status.
- The `onlyEditor` modifier ensures that certain critical functions can only be called by an editor.
- Editor-specific functions include:
  - `addEditor(address)`
  - `removeEditor(address)`
  - `setNFTWhitelisted`
  - `setProfitCollector`
  - `setListingFeeBps`, `setAuctionFeeBps`
  - `vaultTransferOut` and `vaultTransferOutBatch`
  - `listVaultToken`, `massListVaultTokens`
  - `massAuctionVaultTokens`, `createVaultAuction`
  - etc.

3.2 Whitelist

- The marketplace only works with NFTs from **whitelisted** contracts.
- `mapping(address => bool)public whitelistedNFTs;` tracks which NFT contracts are allowed.
- The `setNFTWhitelisted(address,bool)` function (editor-only) toggles or sets whitelisted status.
- In `onERC721Received`, the contract checks `require(whitelistedNFTs[msg.sender])`, ensuring only approved ERC721s can be deposited.

3.3 Profit Collector

- `profitCollector` is an address that receives proceeds from:
  - Sales of NFTs owned by the vault (i.e., `seller == address(this)`).
  - Listing and auction fees from user-owned items.
- `setProfitCollector` can update this address (editor-only).

3.4 Fee Parameters

- `listingFeeBps` and `auctionFeeBps` store the fee in **basis points**, where 100 points = 1%.
- By default set to 100 (1%), but an editor can change them with:
  - `setListingFeeBps(uint256 newFee)`
  - `setAuctionFeeBps(uint256 newFee)`
- Must never exceed 10000 (100%).

3.5 Vault Storage and Transfer

- If an NFT is **safe transferred** to the contract, `onERC721Received` triggers.
- The NFT is stored in a `VaultItem` struct array `vaultItems`, and the mapping `vaultIndex [nftContract][tokenId]` indicates where it sits in that array.
- The vault is important because the contract can **mass list** or **mass auction** NFTs that it owns, or an editor can **transfer them out** via `vaultTransferOut`.
- The function `vaultTransferOut` checks the vault state, marks the NFT as removed, and calls `safeTransferFrom`.
- If multiple NFTs must be sent, `vaultTransferOutBatch` can handle them in a single transaction, respecting `maxVaultBatchTransferSize`.

3.6 Listings (Fixed Price)

- Two scenarios for listings:
  1. **Vault-owned listing**: Created by an editor for an NFT in the vault. No listing fee applies. If sold, the proceeds go to `profitCollector`.
  2. **User-owned listing**: A user calls `listToken` with their NFT. A listing fee fee = (price × listingFeeBps)/10000 must be paid up front. The NFT is transferred into the contract. If sold, proceeds go back to the user.
- The `Listing` struct holds `seller`, `nftContract`, `tokenId`, `price`, and `active`.
- Buying an NFT uses `buyToken`. The buyer sends exactly `price` in msg.value.
- The function `massListVaultTokens` can list every vault item in a single call at `defaultVaultListingPri` .

3.7 Auctions

- The marketplace supports simple auctions with:
  - **initialPrice** (the minimum opening bid).
  - **minStep** (minimum increment above the highest bid).
  - **maxBids** (the auction ends after exactly `maxBids` valid bids).
- The `Auction` struct stores relevant fields. The contract uses `auctions[nftContract][tokenId]` to manage each auction.
- `createVaultAuction` for vault-owned NFTs (no listing fee). `createAuction` for user-owned NFTs (requires fee = (initialPrice × auctionFeeBps)/10000).
- Bidding:
  - If `bidCount == 0`, the next valid bid ≥ initialPrice.
  - Otherwise, must be ≥ (highestBid + minStep).
  - The old highest bidder is immediately refunded, leaving only one bidder's funds locked at a time.
  - When bidCount ≥ maxBids, the auction ends automatically by calling `_endAuctionInternal`.
- `endAuction` can end if bidCount ≥ maxBids or if an editor chooses to forcibly finalize.
- `cancelAuction` allows the seller (or editor, if from vault) to withdraw the auction, refunding the highest bidder if any, and returning the NFT to the seller.
- Final proceeds for vault auctions go to `profitCollector`. For user auctions, the user receives the funds.

4 Security and Usage Notes

- The contract inherits `ReentrancyGuard` from OpenZeppelin, preventing re-entrant calls on sensitive functions such as `buyToken` or `placeBid`.
- Only whitelisted NFT contracts can be deposited into the vault or listed by users, mitigating the risk of non-standard or malicious tokens.
- The `maxVaultBatchTransferSize` ensures that in a single transaction, the editor won't exceed a certain number of NFTs, which can prevent overly large loops in a single call.
- The listing and auction fees are configurable by an editor up to 100% (10000 bps).

5 Conclusion

The `NFTMarketplace` contract is a versatile ERC721 marketplace that enables both simple fixed-price listings and counted-bid auctions, with an administrative editor role, a vault system, adjustable fees, and built-in safe transfer logic for secure NFT handling. The code demonstrates a typical flow:

1. NFTs are transferred in (automatically recognized in the vault).
2. Editors can mass list or mass auction these vault items.
3. Users can list or auction their own NFTs (paying a configurable listing fee).
4. Buyers pay the exact asking price, or place bids in auctions with immediate refunds to outbid participants.
5. Final proceeds are distributed to the `profitCollector` if from the vault, or to the original user if user-owned.