onemind

ONEMIND

SMART CONTRACT AUDIT

zokyo

October 18th 2022 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

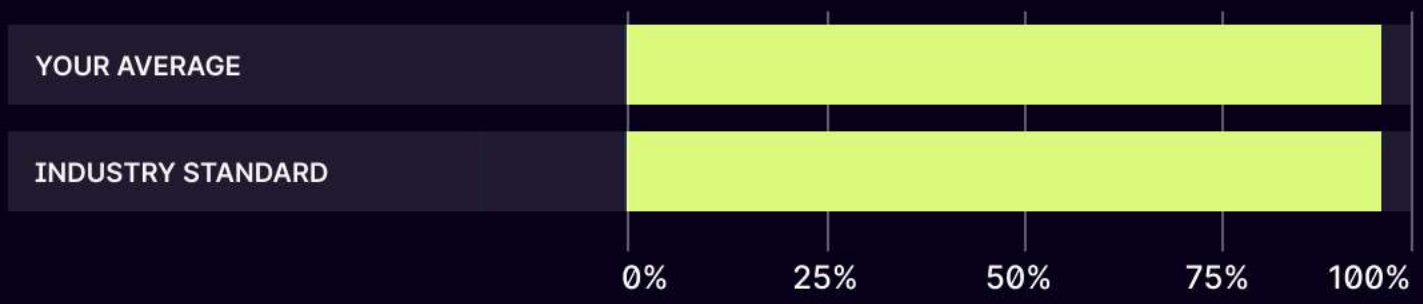
TECHNICAL SUMMARY

This document outlines the overall security of the OneMind smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the OneMind smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The 95% of the code is testable, which corresponds the standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the OneMind team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the OneMind repository.
https://github.com/monterail/tokenyard-evm

Initial commit: b5c2dc9fe32c3da627135e2131330cee11a9c4a9

Final commit: 4368f21c18a080ae90f92d8b7766f9746ea57dc1

Within the scope of this audit, Zokyo auditors have reviewed the following contract(s):

- abstractions/FeePayer.sol
- abstractions/Maintainable.sol
- AuctionFactory.sol
- CollectionFactory.sol
- DutchBid.soll
- NFTCollectionERC721.sol
- NFTCollectionERC1155.sol
- PlatformAuction.sol
- SimpleSale.sol
- StandardBid.sol
- Vault.sol

Throughout the review process, Zokyo Security ensures that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of OneMind smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented any issues as they were discovered. A part of this work included writing a unit test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.	04	Thorough manual review of the codebase, line by line.

Executive Summary

During the audit, Zokyo Security has carefully checked all the contracts provided by the OneMind team. OneMind protocol consists of ERC721 and ERC1155 implementations, 3 different auction contracts, factories for creating NFTs, Auction contracts and utilities. Therefore, the goal of the audit was to ensure the safety of the NFT and Auction contracts, as well as users' funds stored in the Auction, validate the code of the contracts against the list of common security vulnerabilities, verify that the best Solidity practises are applied in terms of security and gas optimization.

The audit process was splitted into two parts: manual and automatic testing. There were 3 high-severity issues connected to ETH transferring that we found during the manual audit. These issues could have led to the possible stuck ETH on contract's balance, which couldn't be rescued, especially considering the fact that the contracts are not upgradable. Other issues were connected to missing parameters validation, unchecked ERC20 tokens transfer, and logic validation. All of these issues were successfully fixed by the OneMind team.

The project provided by the OneMind team contained unit-tests as well. Nevertheless, Zokyo Security has prepared our own set of unit tests to ensure that the logic of the contracts works as intended and all the funds and NFT tokens stored on contracts' balances are safe.

The overall security of the protocol is high enough. The contracts are not upgradable, well-written and have a good natspec documentation. After all the fixes were done, the contracts have passed all security tests.

PROTOCOL OVERVIEW

ONEMIND PROTOCOL

OneMind Protocol is a protocol that allows users to create, buy, and sell NFTs or participate in an NFT auction.
The protocol has 9 main contracts:
- Factories (AuctionFactory.sol, CollectionFactory.sol)
- NFT Collections (NFTCollectionERC721.sol, NFTCollectionERC1155.sol)
- Sale Logic (PlatformAuction.sol ⇒ DutchBid.sol, StandardBid.sol, SimpleSale.sol)
- Vault.sol

OneMind Protocol allows users to:
- Create an NFT collection (CollectionFactory.sol)
- Create an NFT auction (AuctionFactory.sol)
- Buy, sell, or bid on NFTs (DutchBid.sol, StandardBid.sol, SimpleSale.sol)

NFTCollectionERC721.sol is a contract that inherits basic ERC721 contract with additional IERC2981 implementation for royalties.

NFTCollectionERC1155.sol is a contract that inherits basic ERC1155 contract with additional IERC2981 implementation for royalties.

AuctionFactory.sol is a contract that creates NFT auctions using onERC721Received() and onERC1155Received() functions. By using this functions, the user can send their token to the contract, and the auction will begin.

CollectionFactory.sol is a contract that creates an NFT Collection using create721Collection() and create1155Collection() functions. The user who invoked the function will be the owner of the collection. NFT implementations are created from NFTCollectionERC721 and NFTCollectionERC1155 contracts.

FeePayer.sol is a contract that handles creation of NFTs from collections created by CollectionFactory.sol so that the user does not pay creation fees.

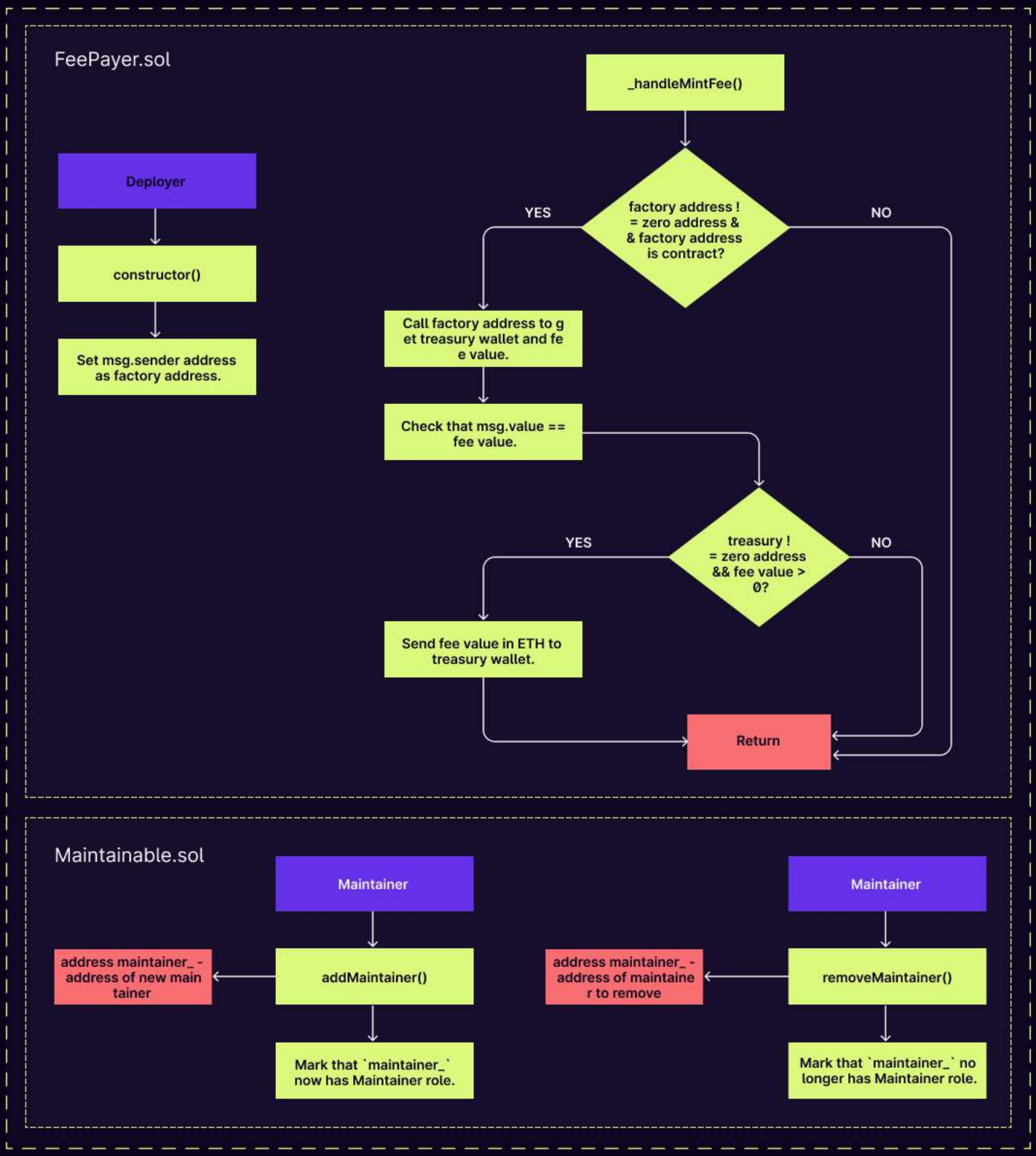
PlatformAuction.sol is the main contract containing key functionality for NFT auctions.

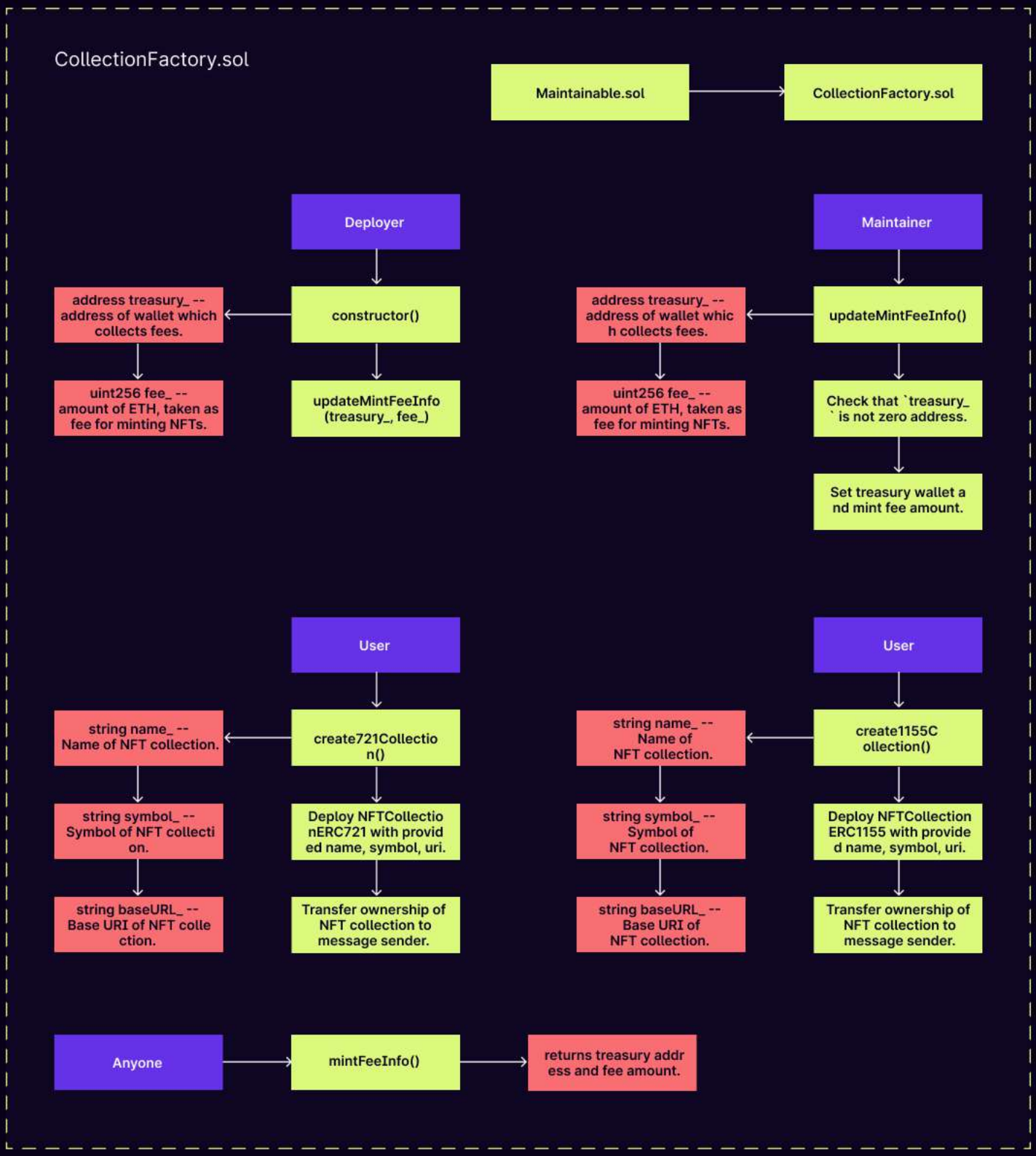
SimpleSale.sol is a contract that inherits PlatformAuction contract. Its main functionality is to end the auction when the user bids on the NFT.

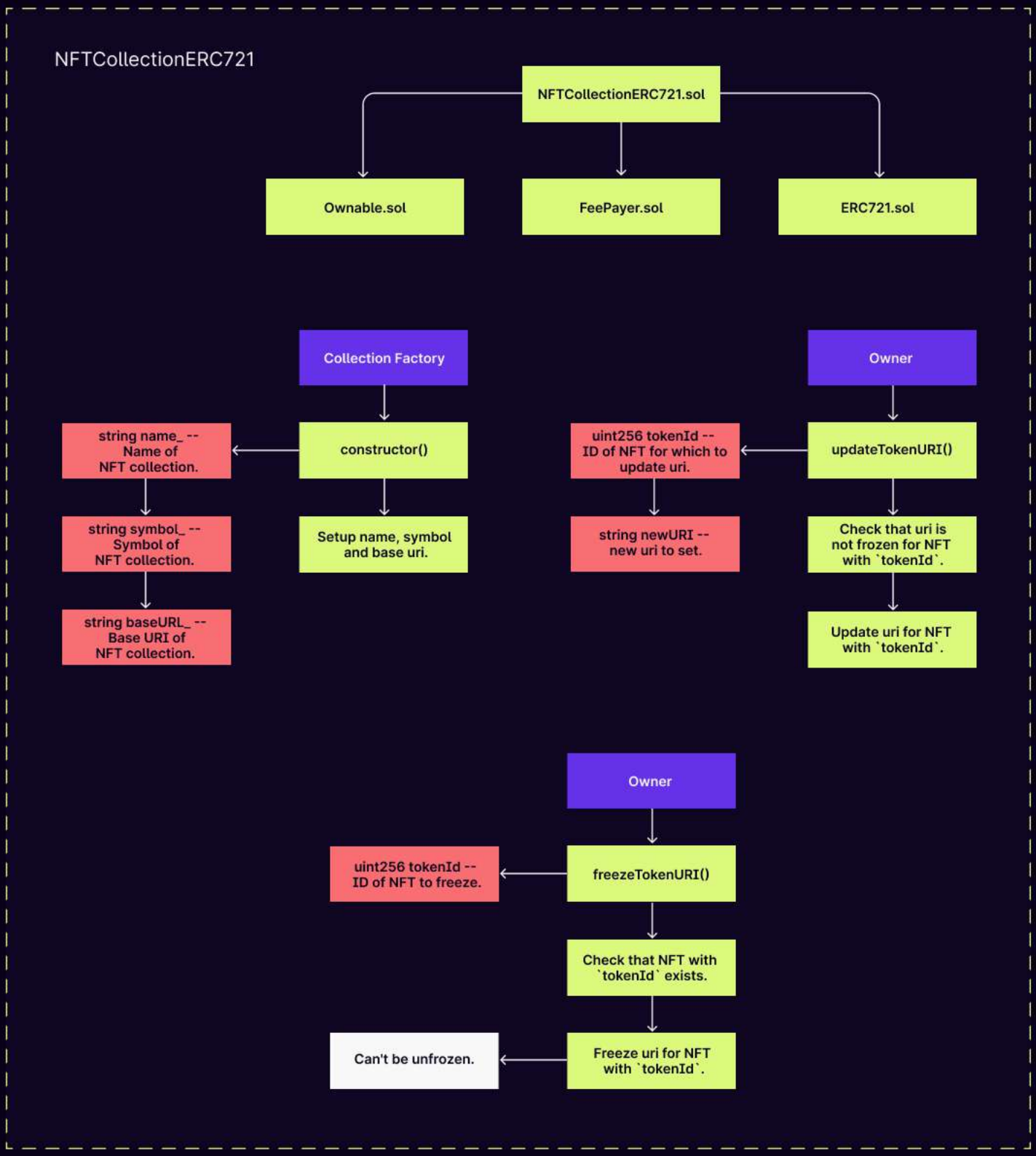
StandardBid.sol is a contract that inherits PlatformAuction contract. Its main functionality is to assign the victory to the highest bid when the auction time ends. Also, the user can use the buyNow() function, which ends the auction and transfers the NFT to the user.

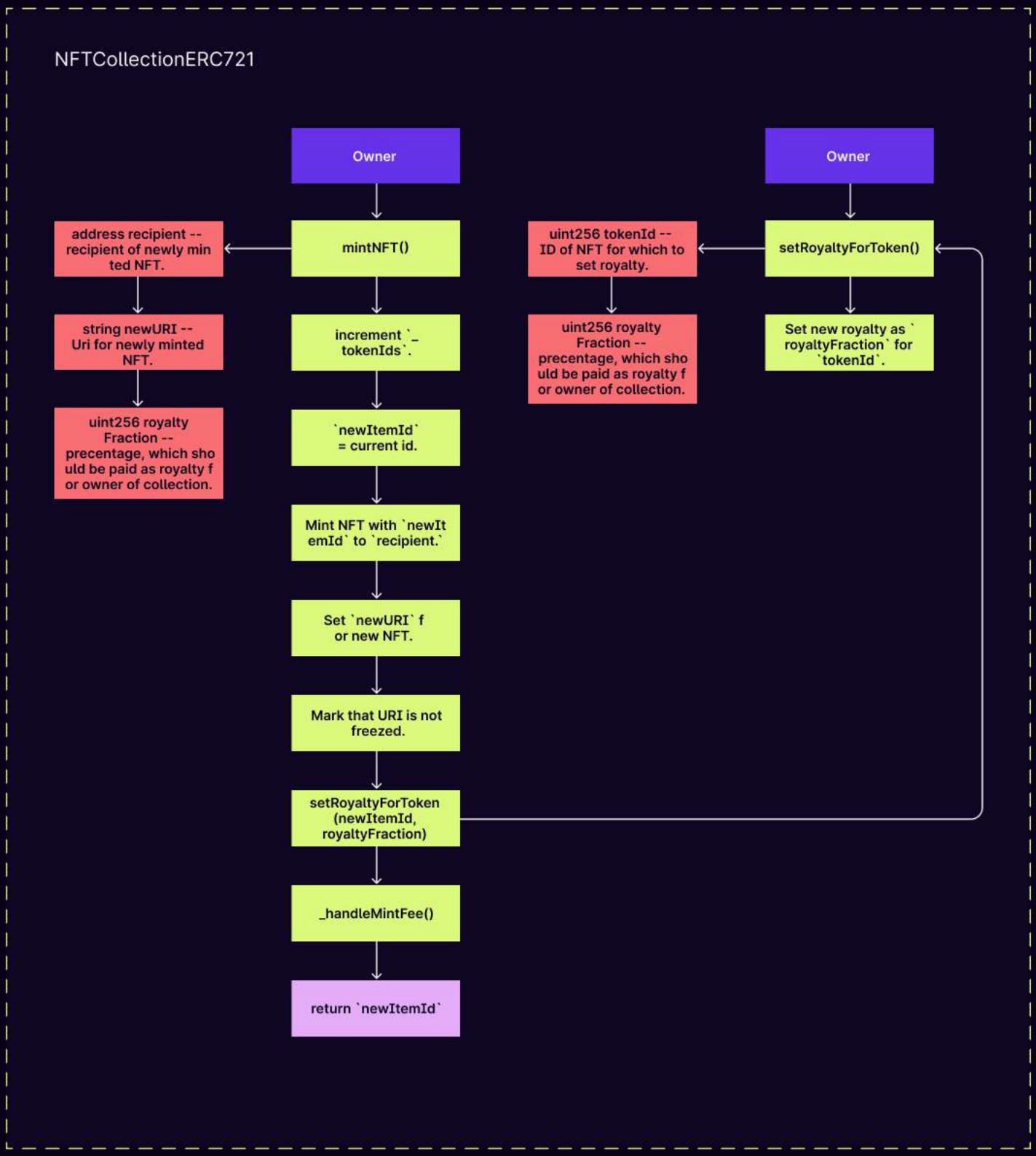
DutchBid.sol is a contract that inherits PlatformAuction contract. This contract implements the Dutch auction system allowing to decrease the price over time and allocating the NFT if the user bids, thus ending the auction.

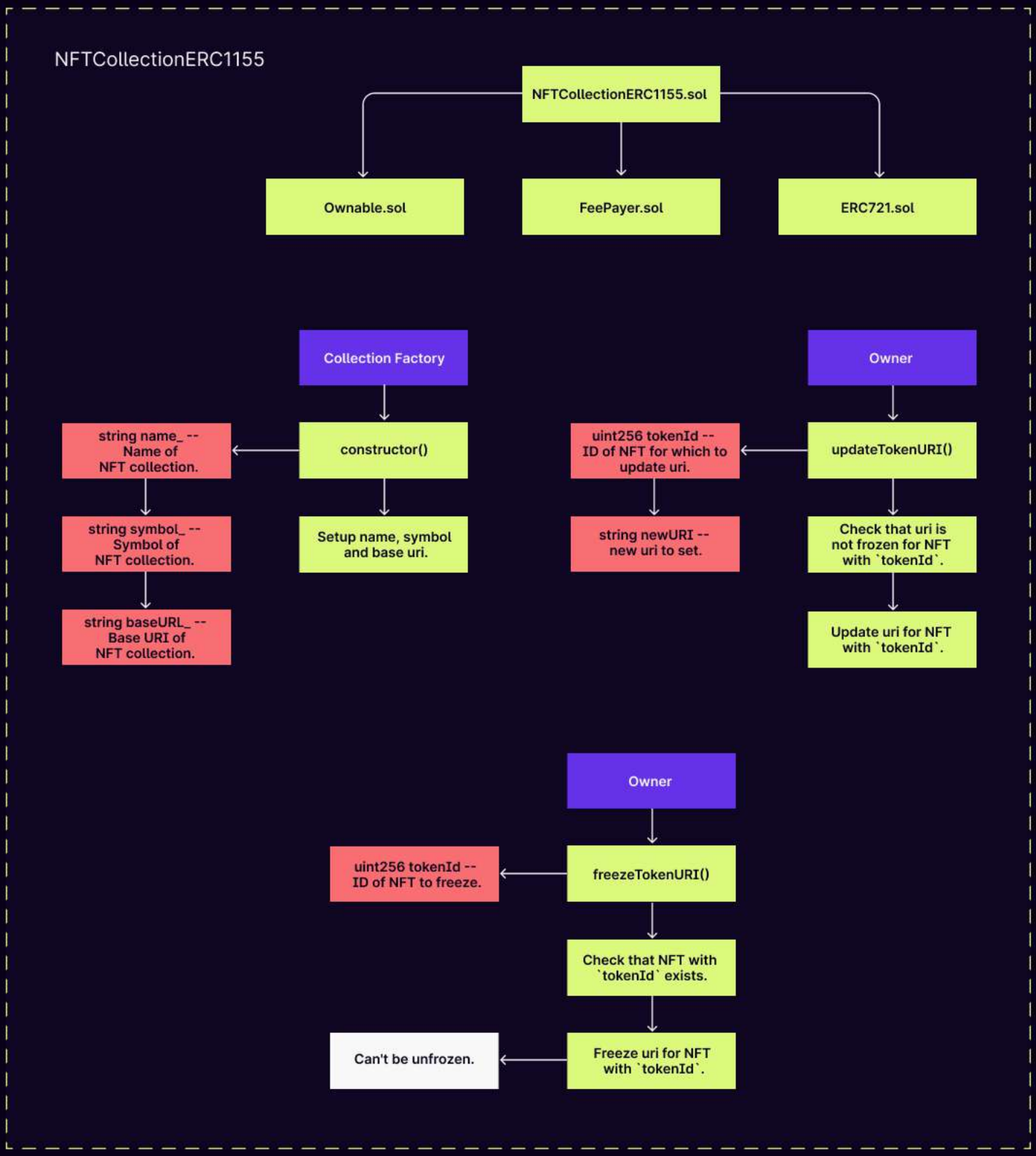
Vault.sol is a contract designed as a multisig wallet that can store and transfer a specified ERC20 token. All the owners of the vault must confirm a transfer transaction, so that it can be executed.

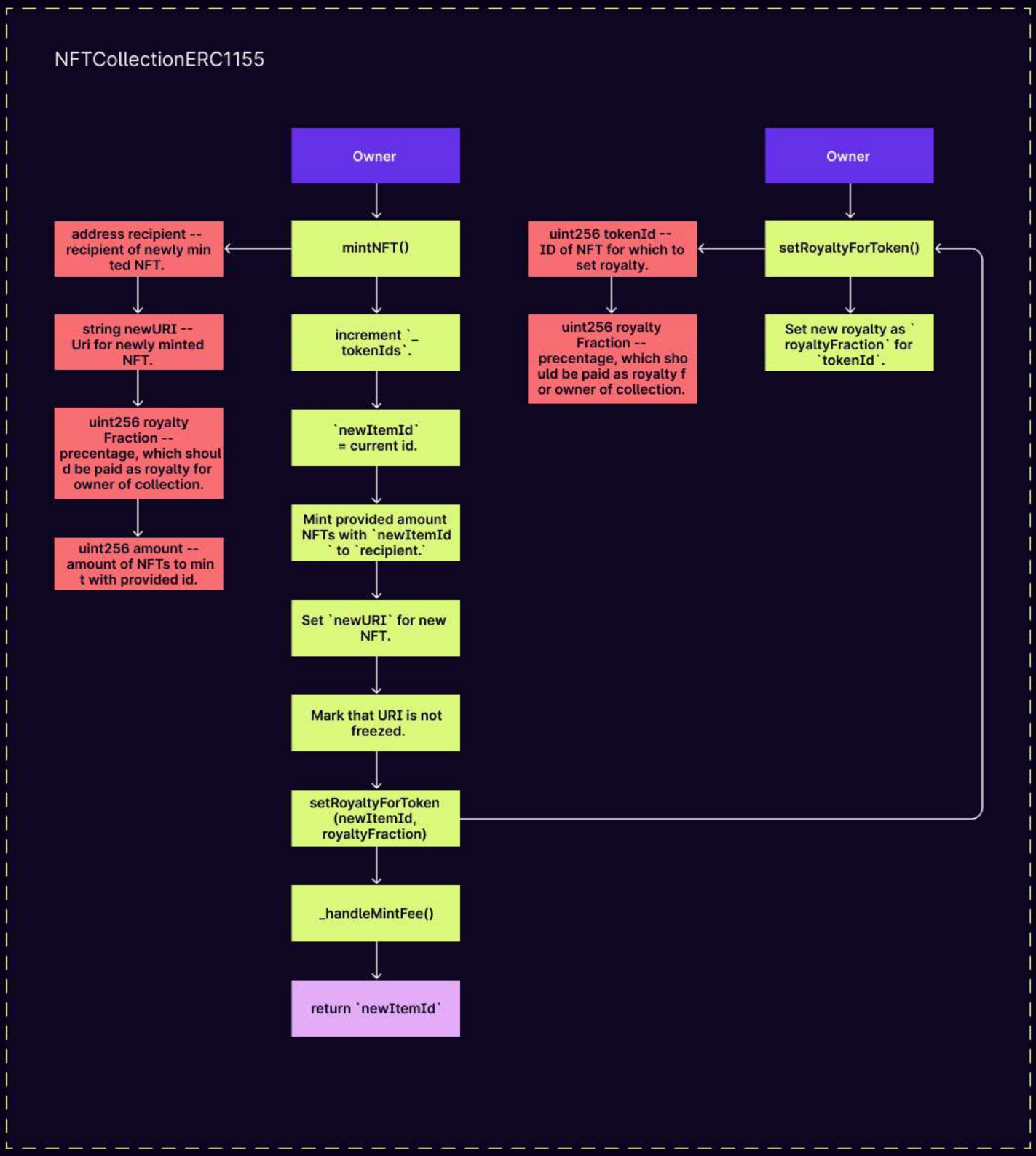

FeePayer.sol
_handleMintFee()
Deployer
constructor()
Set msg.sender address as factory address.
factory address ! = zero address & & factory address is contract?
YES
NO
Call factory address to g et treasury wallet and fe e value.
Check that msg.value == fee value.
treasury ! = zero address && fee value > 0?
YES
NO
Send fee value in ETH to treasury wallet.
Return
Maintainable.sol
Maintainer
address maintainer_ - address of new main tainer
addMaintainer()
Mark that `maintainer_` now has Maintainer role.
Maintainer
address maintainer_ - address of maintaine r to remove
removeMaintainer()
Mark that `maintainer_` no longer has Maintainer role.

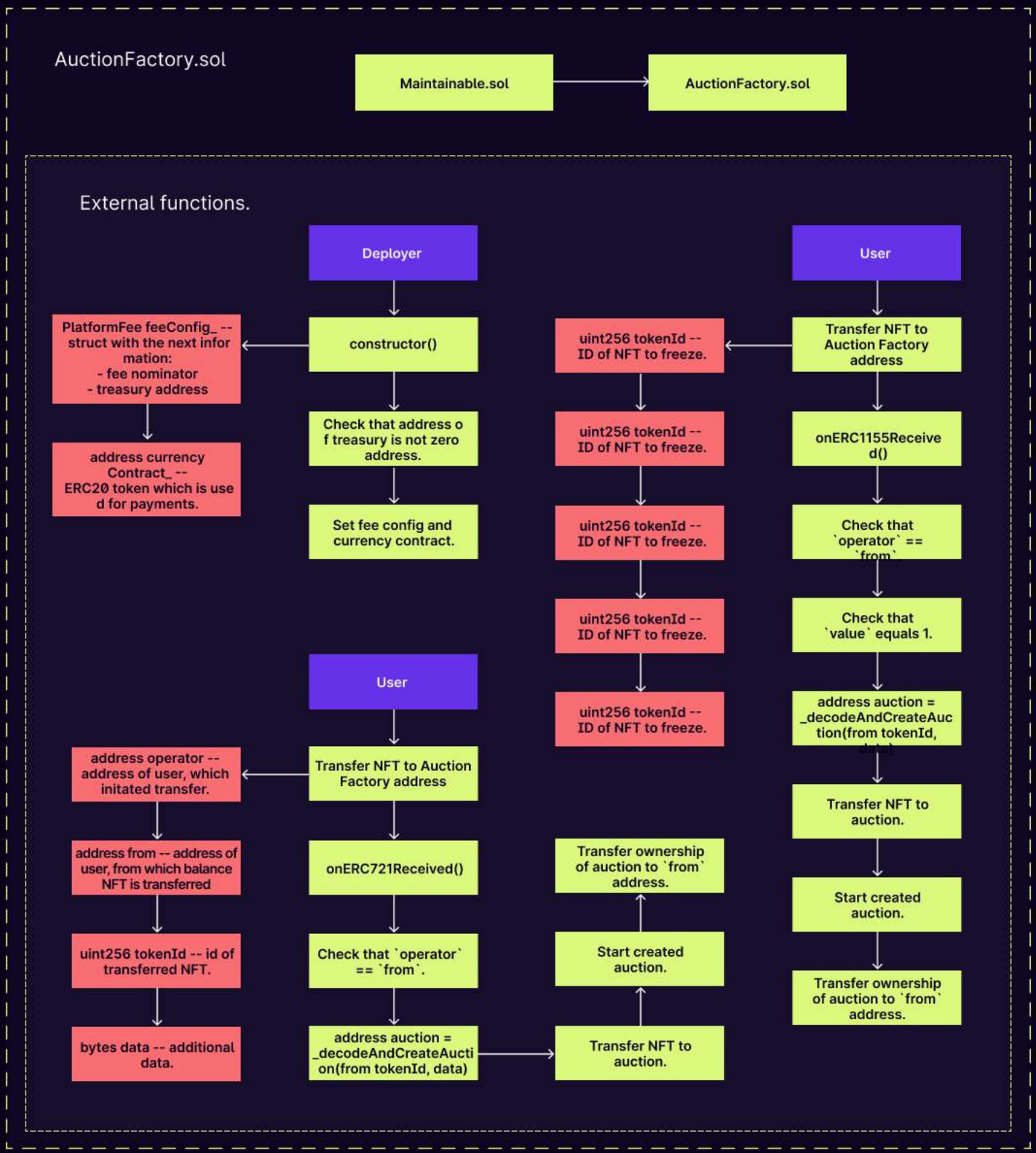

CollectionFactory.sol
Maintainable.sol
CollectionFactory.sol
Deployer
constructor()
address treasury_ -- address of wallet which collects fees.
uint256 fee_ -- amount of ETH, taken as fee for minting NFTs.
updateMintFeeInfo (treasury_, fee_)
Maintainer
updateMintFeeInfo()
address treasury_ -- address of wallet whic h collects fees.
uint256 fee_ -- amount of ETH, taken as fee for minting NFTs.
Check that `treasury_ ` is not zero address.
Set treasury wallet a nd mint fee amount.
User
create721Collectio n()
string name_ -- Name of NFT collection.
string symbol_ -- Symbol of NFT collecti on.
string baseURL_ -- Base URI of NFT colle ction.
Deploy NFTCollectio nERC721 with provid ed name, symbol, uri.
Transfer ownership of NFT collection to message sender.
User
create1155C ollection()
string name_ -- Name of NFT collection.
string symbol_ -- Symbol of NFT collection.
string baseURL_ -- Base URI of NFT collection.
Deploy NFTCollection ERC1155 with provide d name, symbol, uri.
Transfer ownership of NFT collection to message sender.
Anyone
mintFeeInfo()
returns treasury addr ess and fee amount.

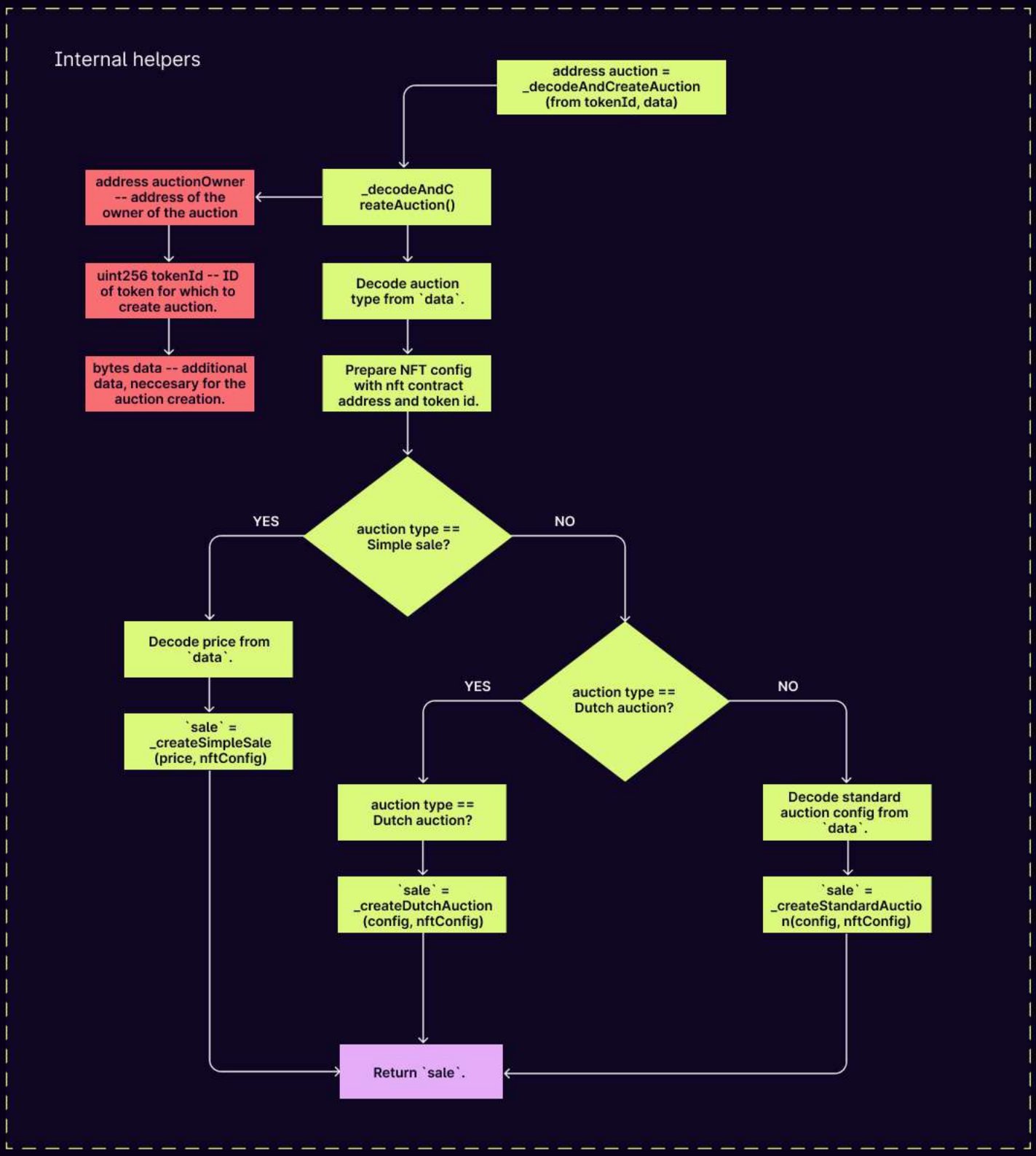

NFTCollectionERC721
NFTCollectionERC721.sol
Ownable.sol
FeePayer.sol
ERC721.sol
Collection Factory
constructor()
string name_ -- Name of NFT collection.
string symbol_ -- Symbol of NFT collection.
string baseURL_ -- Base URI of NFT collection.
Setup name, symbol and base uri.
Owner
updateTokenURI()
uint256 tokenId -- ID of NFT for which to update uri.
string newURI -- new uri to set.
Check that uri is not frozen for NFT with `tokenId`.
Update uri for NFT with `tokenId`.
Owner
freezeTokenURI()
uint256 tokenId -- ID of NFT to freeze.
Check that NFT with `tokenId` exists.
Freeze uri for NFT with `tokenId`.
Can't be unfrozen.


NFTCollectionERC721
Owner
mintNFT()
address recipient -- recipient of newly min ted NFT.
string newURI -- Uri for newly minted NFT.
uint256 royalty Fraction -- precentage, which sho uld be paid as royalty f or owner of collection.
increment `_ tokenIds`.
`newItemId` = current id.
Mint NFT with `newIt emId` to `recipient.`
Set `newURI` f or new NFT.
Mark that URI is not freezed.
setRoyaltyForToken (newItemId, royaltyFraction)
_handleMintFee()
return `newItemId`
Owner
setRoyaltyForToken()
uint256 tokenId -- ID of NFT for which to set royalty.
uint256 royalty Fraction -- precentage, which sho uld be paid as royalty f or owner of collection.
Set new royalty as ` royaltyFraction` for `tokenId`.

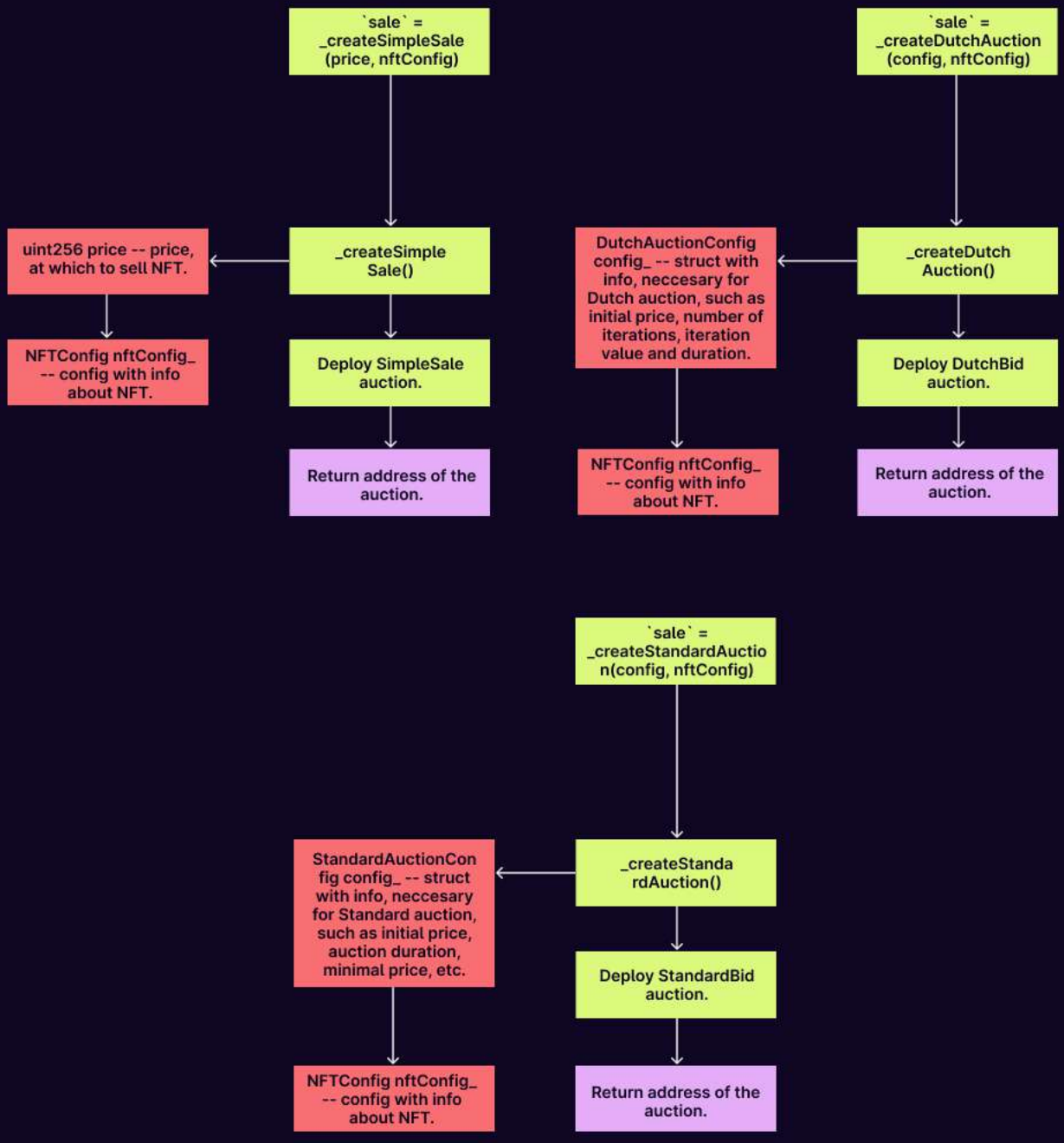

NFTCollectionERC1155
NFTCollectionERC1155.sol
Ownable.sol
FeePayer.sol
ERC721.sol
Collection Factory
constructor()
string name_ -- Name of NFT collection.
string symbol_ -- Symbol of NFT collection.
string baseURL_ -- Base URI of NFT collection.
Setup name, symbol and base uri.
Owner
updateTokenURI()
uint256 tokenId -- ID of NFT for which to update uri.
string newURI -- new uri to set.
Check that uri is not frozen for NFT with `tokenId`.
Update uri for NFT with `tokenId`.
Owner
freezeTokenURI()
uint256 tokenId -- ID of NFT to freeze.
Check that NFT with `tokenId` exists.
Freeze uri for NFT with `tokenId`.
Can't be unfrozen.


NFTCollectionERC1155
Owner
mintNFT()
address recipient -- recipient of newly min ted NFT.
string newURI -- Uri for newly minted NFT.
uint256 royalty Fraction -- precentage, which shoul d be paid as royalty for owner of collection.
uint256 amount -- amount of NFTs to min t with provided id.
increment `_ tokenIds`.
`newItemId` = current id.
Mint provided amount NFTs with `newItemId ` to `recipient.`
Set `newURI` for new NFT.
Mark that URI is not freezed.
setRoyaltyForToken (newItemId, royaltyFraction)
_handleMintFee()
return `newItemId`
Owner
setRoyaltyForToken()
uint256 tokenId -- ID of NFT for which to set royalty.
uint256 royalty Fraction -- precentage, which sho uld be paid as royalty f or owner of collection.
Set new royalty as ` royaltyFraction` for `tokenId`.

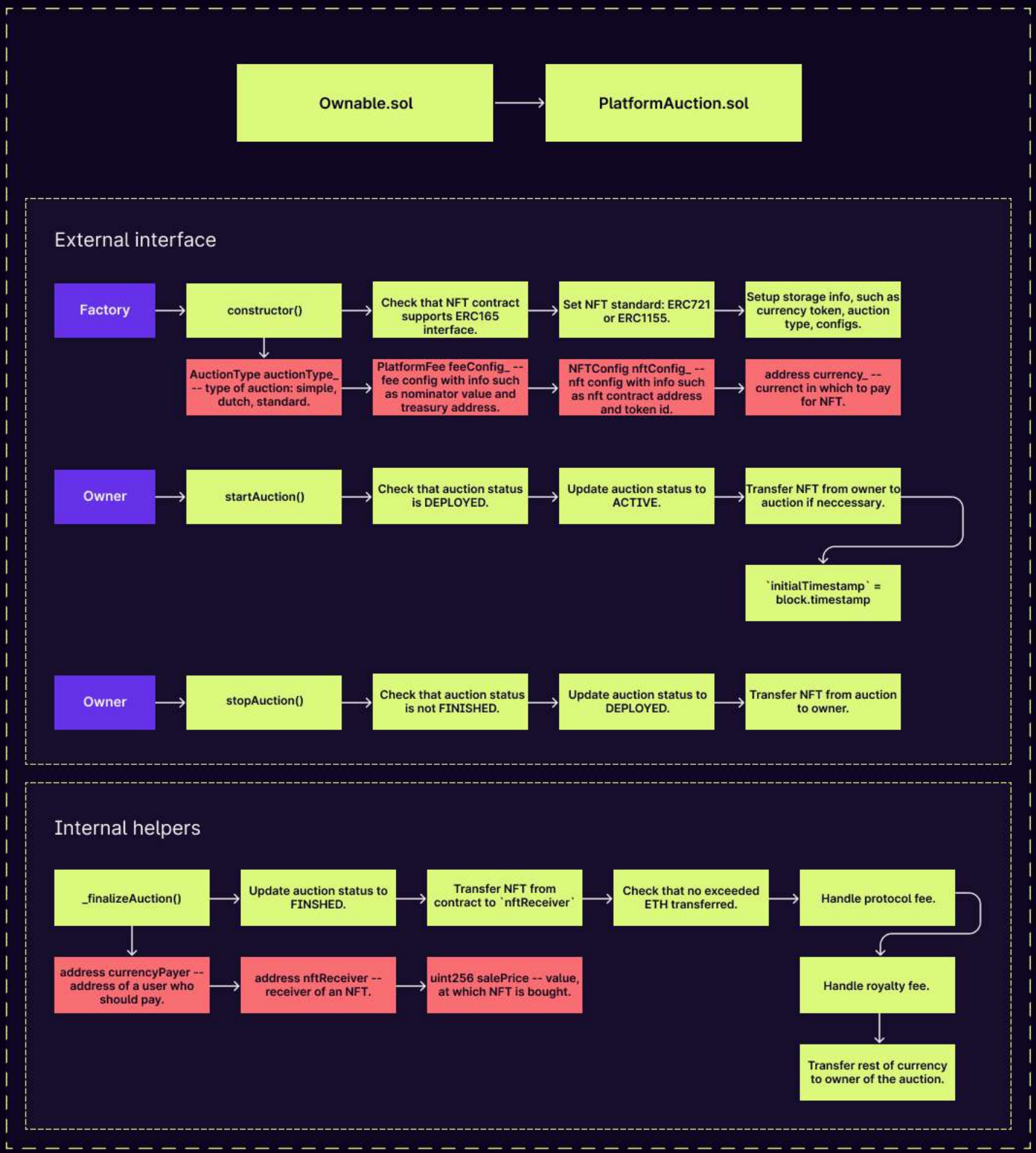

AuctionFactory.sol
Maintainable.sol
AuctionFactory.sol
External functions.
Deployer
constructor()
PlatformFee feeConfig_ -- struct with the next infor mation:
- fee nominator
- treasury address
address currency Contract_ -- ERC20 token which is use d for payments.
Check that address o f treasury is not zero address.
Set fee config and currency contract.
User
Transfer NFT to Auction Factory address
address operator -- address of user, which initated transfer.
address from -- address of user, from which balance NFT is transferred
uint256 tokenId -- id of transferred NFT.
bytes data -- additional data.
onERC721Received()
Check that `operator` == `from`.
address auction = _decodeAndCreateAucti on(from tokenId, data)
uint256 tokenId -- ID of NFT to freeze.
uint256 tokenId -- ID of NFT to freeze.
uint256 tokenId -- ID of NFT to freeze.
uint256 tokenId -- ID of NFT to freeze.
uint256 tokenId -- ID of NFT to freeze.
Transfer NFT to auction.
Start created auction.
Transfer ownership of auction to `from` address.
User
Transfer NFT to Auction Factory address
onERC1155Receive d()
Check that `operator` == `from`
Check that `value` equals 1.
address auction = _decodeAndCreateAuc tion(from tokenId,
Transfer NFT to auction.
Start created auction.
Transfer ownership of auction to `from` address.


Internal helpers
address auction = _decodeAndCreateAuction (from tokenId, data)
_decodeAndCreateAuction()
address auctionOwner -- address of the owner of the auction
uint256 tokenId -- ID of token for which to create auction.
bytes data -- additional data, neccesary for the auction creation.
Decode auction type from `data`.
Prepare NFT config with nft contract address and token id.
auction type == Simple sale?
YES
NO
Decode price from `data`.
`sale` = _createSimpleSale (price, nftConfig)
auction type == Dutch auction?
YES
NO
auction type == Dutch auction?
Decode standard auction config from `data`.
`sale` = _createDutchAuction (config, nftConfig)
`sale` = _createStandardAuction(config, nftConfig)
Return `sale`.

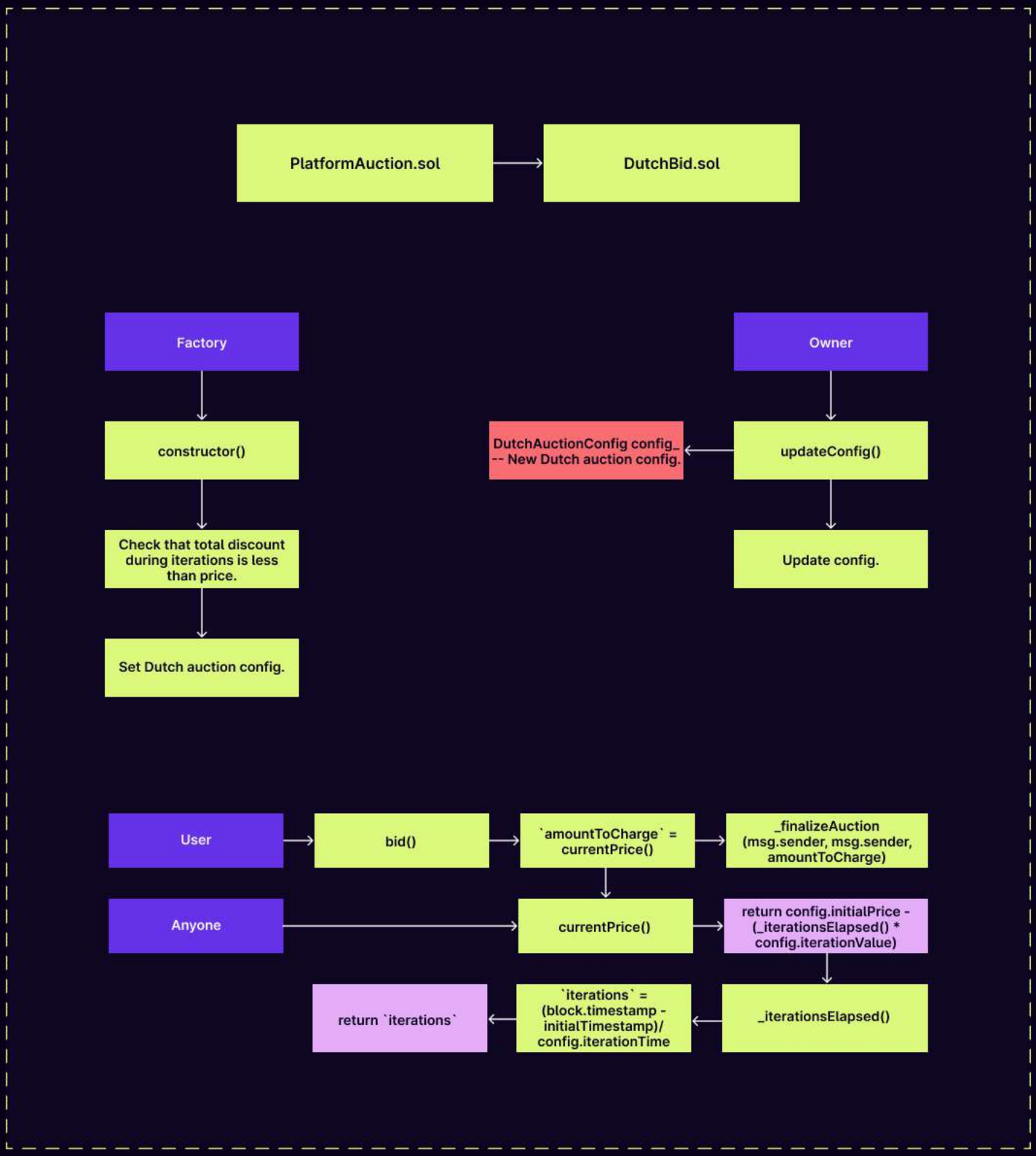
Internal helpers



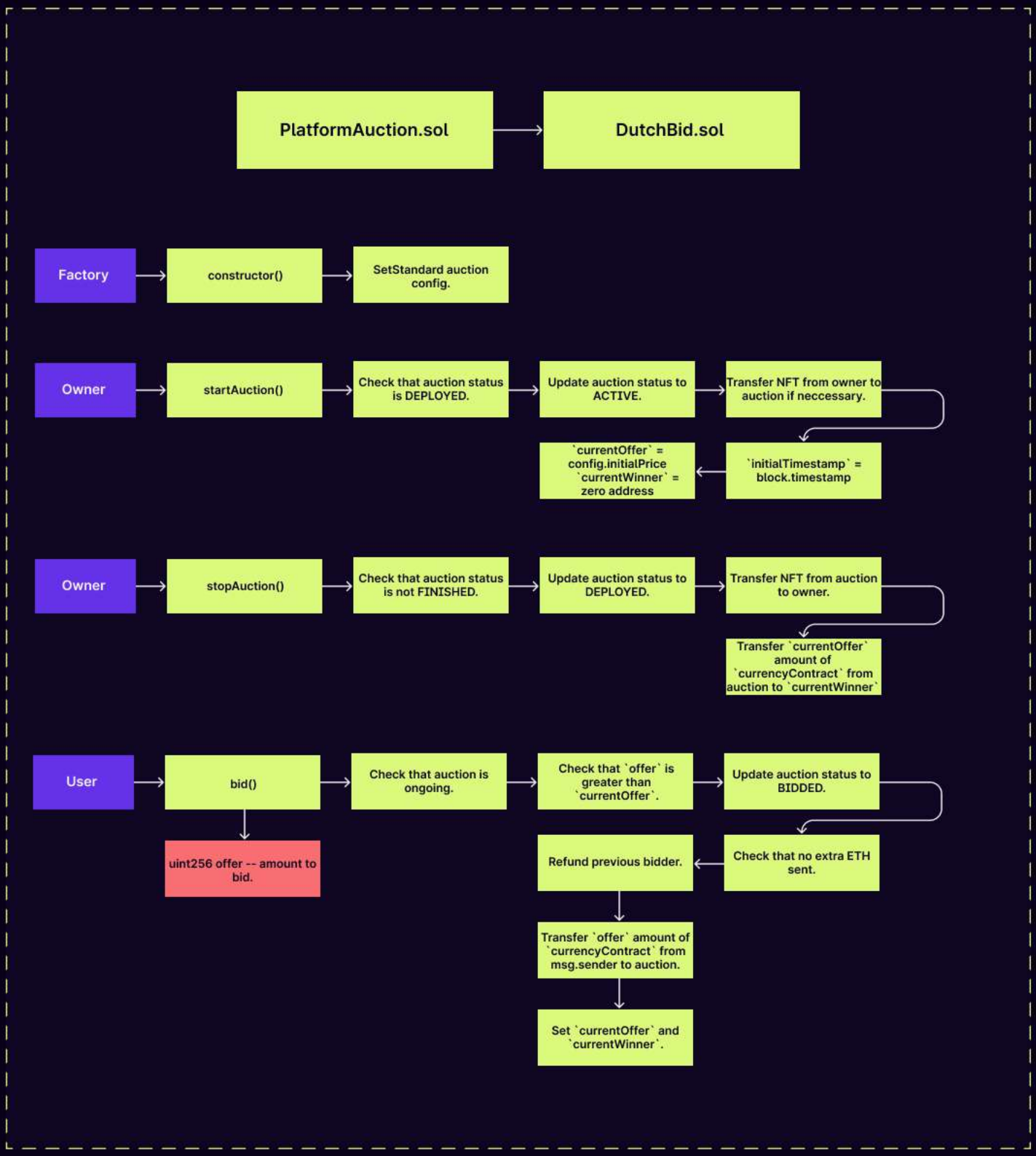
PLATFORMAUCTION.SOL

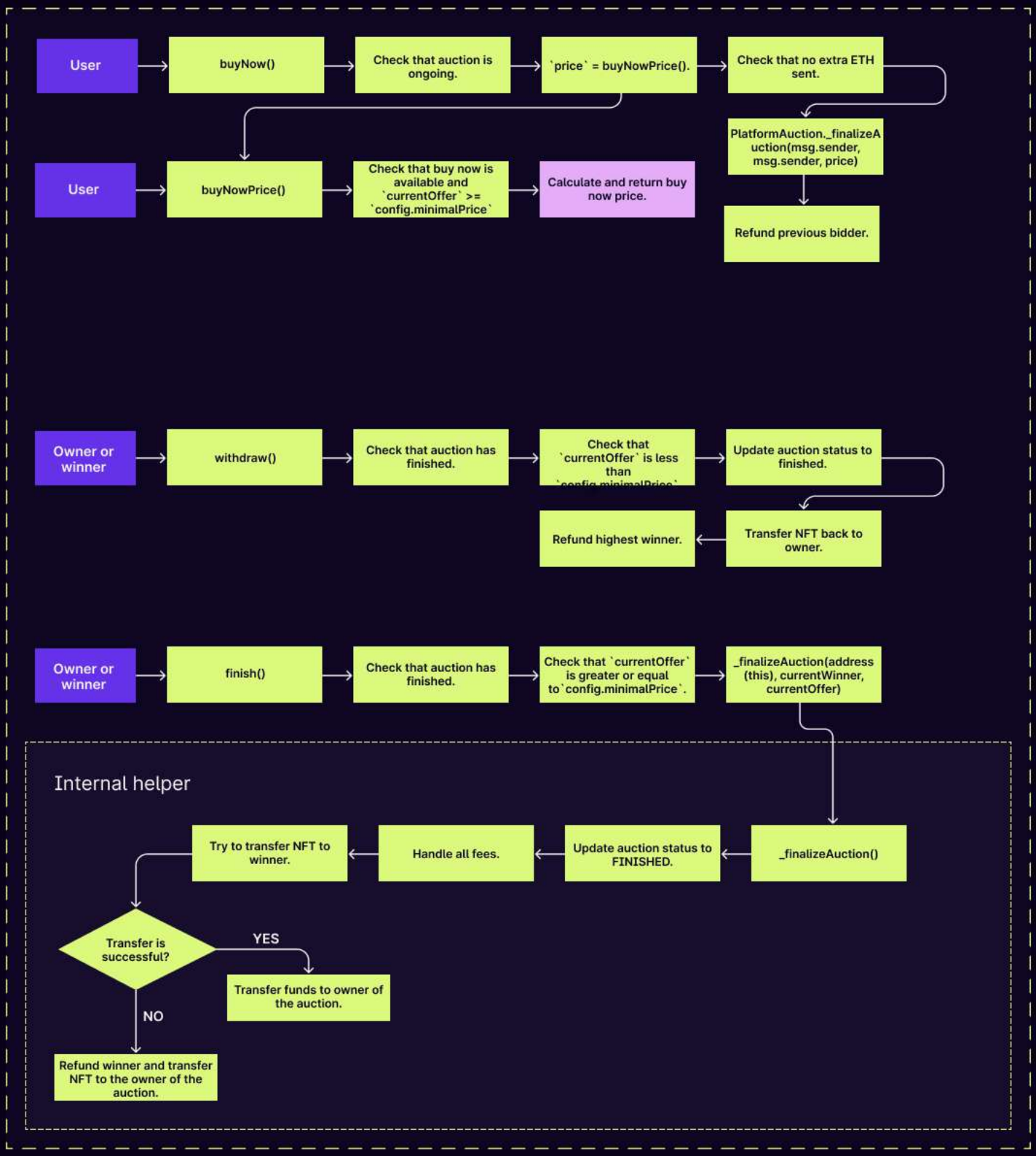


SIMPLESALE.SOL

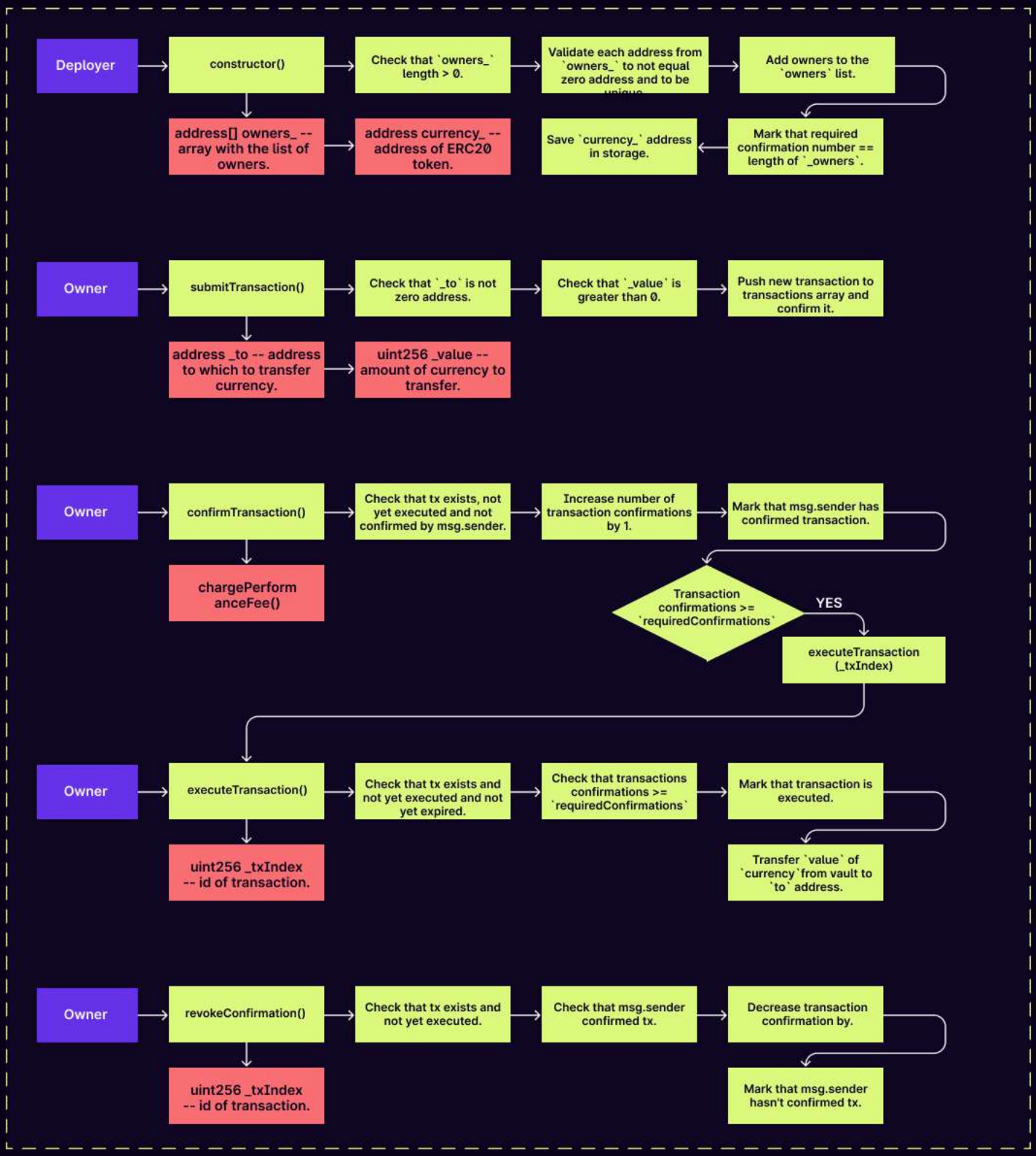

PlatformAuction.sol
DutchBid.sol
Factory
constructor()
Check that total discount during iterations is less than price.
Set Dutch auction config.
Owner
updateConfig()
DutchAuctionConfig config_ -- New Dutch auction config.
Update config.
User
bid()
`amountToCharge` = currentPrice()
_finalizeAuction (msg.sender, msg.sender, amountToCharge)
Anyone
currentPrice()
return config.initialPrice - (_iterationsElapsed() * config.iterationValue)
_iterationsElapsed()
`iterations` = (block.timestamp - initialTimestamp)/ config.iterationTime
return `iterations`

STANDARDBID.SOL




User
buyNow()
Check that auction is ongoing.
`price` = buyNowPrice().
Check that no extra ETH sent.
PlatformAuction._finalizeAuction(msg.sender, msg.sender, price)
Refund previous bidder.
User
buyNowPrice()
Check that buy now is available and `currentOffer` >= `config.minimalPrice`
Calculate and return buy now price.
Owner or winner
withdraw()
Check that auction has finished.
Check that `currentOffer` is less than
Update auction status to finished.
Transfer NFT back to owner.
Refund highest winner.
Owner or winner
finish()
Check that auction has finished.
Check that `currentOffer` is greater or equal to`config.minimalPrice`.
_finalizeAuction(address (this), currentWinner, currentOffer)
Internal helper
_finalizeAuction()
Update auction status to FINISHED.
Handle all fees.
Try to transfer NFT to winner.
Transfer is successful?
YES
Transfer funds to owner of the auction.
NO
Refund winner and transfer NFT to the owner of the auction.

VAULT.SOL


Deployer
constructor()
Check that `owners_` length > 0.
Validate each address from `owners_` to not equal zero address and to be
Add owners to the `owners` list.
address[] owners_ -- array with the list of owners.
address currency_ -- address of ERC20 token.
Save `currency_` address in storage.
Mark that required confirmation number == length of `_owners`.
Owner
submitTransaction()
Check that `_to` is not zero address.
Check that `_value` is greater than 0.
Push new transaction to transactions array and confirm it.
address _to -- address to which to transfer currency.
uint256 _value -- amount of currency to transfer.
Owner
confirmTransaction()
Check that tx exists, not yet executed and not confirmed by msg.sender.
Increase number of transaction confirmations by 1.
Mark that msg.sender has confirmed transaction.
chargePerformanceFee()
Transaction confirmations >= `requiredConfirmations`
YES
executeTransaction (_txIndex)
Owner
executeTransaction()
Check that tx exists and not yet executed and not yet expired.
Check that transactions confirmations >= `requiredConfirmations`
Mark that transaction is executed.
uint256 _txIndex -- id of transaction.
Transfer `value` of `currency` from vault to `to` address.
Owner
revokeConfirmation()
Check that tx exists and not yet executed.
Check that msg.sender confirmed tx.
Decrease transaction confirmation by.
uint256 _txIndex -- id of transaction.
Mark that msg.sender hasn't confirmed tx.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that it can lead to a significant loss, funds may be lost or allocated incorrectly.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

ETH can get stuck on the contract.
FeePayer.sol: function _handleMintFee(), line 24.

The returned value of call() is not checked. In case of the failed ETH transfer, there is no exception and the transaction doesn not revert. Any transferred ETH gets stuck on the contract's balance.

Also, in case the call of `IfeeOracle(_factoryAddress).mintFeeInfo()` fails, there is no exception as well, and ETH stays on contract. Though such a case is possible only when the deployer of the contract is not CollectionFactory.sol, it might still be possible that ETH will get stuck in such a scenario.

Recommendation:
Validate the returned value of call() to be true.

Post-audit:
The returned value of call() is checked now. Also, in case `IfeeOracle(_factoryAddress).mintFeeInfo()` fails, it is validated that msg.value equals `0` so that no ETH gets stuck.

Deprecated ETH transfer.
PlatformAuction.sol: function _currencyTransfer().

Due to the Istanbul update, there were several changes provided to the EVM, which made .transfer() and .send() methods deprecated for the ETH transfer.

It is highly recommended to use .call() functionality with mandatory result check or the built-in functionality of the Address contract from OpenZeppelin library.

Recommendation:
Correct ETH sending functionality with the call() function.

Post-audit:
The deprecated ETH transfer function was replaced with the call() function.

Royalties might get stuck on the contract.
PlatformAuction.sol: function _handleRoyalties(), _currencyTransfer().

There is a check in the _currencyTransfer() function that the `receiver` address is not zero address. In case the owner of the NFT collection has renounced the role with standard Ownable functionality, the _calculateRoyalty() function called within the _handleRoyalties() function will return zero address. This way, the royalty value will be calculated and subtracted from the initial `price` (Line 150), but it won't be transferred and will get stuck on the auction contract's balance.

A similar situation can also happen in the _handleFee() function in case `feeConfig.treasury` is set as zero address.

Recommendation:
Do not subtract the fee value from the price in case the fee value wasn't transferred because of zero address.

Post-audit:
Paid `royalty` amount is now subtracted from `price` in case of a successful royalty transfer. However, in _currencyTransfer(), the `result` flag is initially equal to true. Due to this, the result of the transaction will be returned as true even though, in fact, royalty might not have been transferred.

Post-audit:
The correct result is returned now and no royalties can get stuck.

MEDIUM-1 | RESOLVED

Unchecked transfer.
PlatformAuction.sol: function _currencyTransfer().

Vault.sol: function executeTransaction().

PlatformAuction and Vault are used with regular transfer() and transferFrom() methods with no result check. The return value of an external transfer/transferFrom call is not checked, which could prevent a free deposit/money loss for the user. In this case, it is better to use SafeERC20 library from OpenZeppelin. SafeERC20 has a built-in mandatory checks on any transfer, including non-standard implementations of ERC20 tokens (e.g., USDT token).

Recommendation:
Use SafeERC20 library.

LOW-1 | RESOLVED

Parameters lack validation.
NFTCollectionERC721.sol: function mintNFT(), updateTokenURI().

NFTCollectionERC1155.sol: function mintNFT(), updateTokenURI().

PlatformAuction.sol: constructor().

StandardBid.sol: constructor(), updateConfig().

Vault.sol: constructor().

Function parameters should be validated before setting them in storage.

1) NFTCollectionERC721.sol:

The `recipient` parameter should be validated not to be zero address.

The `newURI` parameter should be validated not to be an empty string.

The `royaltyFraction` parameter should be validated not to exceed `10000`. (Divider from line 32 for ERC721 and line 56 for ERC1155).

2) NFTCollectionERC1155.sol:

all the previous parameters should be validated as well. Additionally, in the

mintNFT() function, the `amount` parameter should be validated in order to be greater than `0`.

3) PlatformAuction.sol:

The `feeConfig_.nominator` parameter should be validated not to exceed `10000` (Divider from line 119).

The `feeConfig_.treasury` parameter should be validated not to be equal to zero address.

4) StandardBid.sol:

The `config_.initialPrice` parameter should be validated in order to be less than `config_.minimalPrice`. Both variables should be validated not be equal to `0`.

The `config_.auctionDuration` parameter should be validated in order to be greater than `0`.

The `config_.buyNowPercent` parameter should be validated in order to be greater than `10000` (a divider in line 127).

5) Vault.sol:

The `currency_` parameter should be validated in order not to be equal to zero address.

Recommendation:
Validate function parameters.

LOW-2 | RESOLVED

Config parameters are not validated.
DutchBid.sol: function updateConfig().

Values from the `config_` parameter are not validated in a way they are validated in the constructor (Line 25). In this case, the owner can set values in a way that the price will go below zero.

Recommendation:
Verify `config_` values like in the constructor.

INFO-1 | RESOLVED

Contracts can be left without maintainers.
Maintainers.sol: function removeMaintainer().

Users with the maintainer role can remove other maintainers, including themselves. In this case, a malicious maintainer can remove all other maintainers and leave the contract without users with such a role. The issue is marked as informational since initially the deployer of the contract is assigned the maintainer role and should grant this role only to validated addresses.

Recommendation:
Verify that the contract can't be left without maintainers.

Post-audit:
The validation was added. Now, the maintainer can't remove other mainteiners.

INFO-2 | RESOLVED

The owner can set URI for a non-existing token.
NFTCollectionERC721.sol, NFTCollectionERC1155.sol: function updateTokenURI().

It is not validated if the provided `tokenId` exists, thus, the owner can modify non-exisiting tokens. The issue is marked as low since URI for non-existing tokens is overridden during the creation of the NFT token.

Recommendation:
Verify that the provided `tokenId` exists.

INFO-3 | RESOLVED

Storage constant should be used.
NFTCollectionERC721.sol: function royaltyInfo(), line 32.

NFTCollectionERC1155.sol: function royaltyInfo(), line 56.

PlatformAuction.sol: function _calculateFee(), line 118.

StandardBid.sol: function buyNowPrice(), line 127.

In order to increase the readability of the contract's code, the value `10000` should be moved to the storage constant.

Recommendation:
Use storage constant.

INFO-4 | RESOLVED

Auctions can be created with any ERC721 and ERC1155 contracts.
AuctionFactory.sol

There are no restrictions for selling NFT contracts on the platform. Thus, malicious NFT contracts can be created and used by malefactors. The issue is marked as informational in case this is an intended functionality to trade any NFTs, not only the ones created through CollectionFactory.sol. Nevertheless, such functionality should be mentioned in report.

Recommendation:
Add a whitelist of NFTs that can be traded or verify that any NFT might be sold.

Post-audit:
According to the OneMind team, it is an intended behavior. Any NFTs that were created not by CollectionFactory will not be listed and displayed on the platform.

INFO-5 | RESOLVED

Possible Denial-of-Service attack.
StandardBid.sol

The contract performs a refund of ETH funds (in case the contract currency is ETH) in all core functions of the contract. In case `currentWinner` is a malicious contract, it could prevent refunding with malicious code in its receive() function. Currently, the issue is marked as informational since the send() function is used to transfer ETH, which doesn't revert on failure. However, such an approach is not considered to be correct. Which is why, it is recommended to refund funds separately from core functions.

Recommendation:
Refund funds aside from core functions.

Post-audit:
It is validated that in case a refund fails, the transaction is not reverted and tokens can be withdrawn later.

INFO-6 | RESOLVED

Transactions are not limited in time.
Vault.sol

Usually, transactions in multi-sig contracts have a deadline, after which the transaction can't be executed. This is done to mitigate the situation when a part of owners iscompromised. For Vault.sol, such an issue is marked as informational since all the owners have to confirm the transaction before it can be executed.

Recommendation:
Add a deadline for transactions or verify that such logic is not necessary.

Post-audit:
All transactions have an expiration timestamp now.

INFO-7 | RESOLVED

The initiator of the transaction has to explicitly confirm it.
Vault.sol: function submitTransaction().

Usually, in multi-sig contracts, submitting a transaction means that msg.sender also confirms it. Currently, in order to confirm a transaction, the owner has to execute one more function. In order to reduce gas spendings for the owner, it is recommended to mark that he confirmed transactions during submitting.

Recommendation:
Mark that msg.sender who calls submitTransaction() confirms it or verify that such logic is not desired for the contract.

Post-audit:
Transactions are automatically confirmed by the owner who has submitted them.

	abstractions/FeePayer.sol	abstractions/Maintainable.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	AuctionFactory.sol	CollectionFactory.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	DutchBid.sol	NFTCollectionERC721.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	Vault.sol	StandardBid.sol	SimpleSale.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	NFTCollectionERC1155.sol	PlatformAuction.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the OneMind team

As a part of our work assisting the OneMind team in verifying the correctness of their contract code, we have checked the full set of unit tests prepared by the OneMind team.

It needs to be mentioned that the original code has a significant original coverage with testing scenarios provided by the OneMind team. All of them were also carefully checked by the team of auditors.

```
AUCTIONS
  Simple Sale
    Group ERC721 | FakeWBTC
      Case 1
✓ Should refuse to start sale (NFT not approved) (44ms)
✓ Should start the sale (892ms)
✓ Should update the price (217ms)
✓ Should stop the sale and transfer NFT back (154ms)
✓ Should refuse to process the transcation (auction stopped) (58ms)
✓ Should restart the sale (115ms)
✓ Should process the transaction (348ms)
      Case 2 - wrong currency type
✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (67ms)
    Group ERC721 | ETH native
      Case 1
✓ Should refuse to start sale (NFT not approved) (48ms)
✓ Should start the sale (141ms)
✓ Should update the price (50ms)
✓ Should stop the sale and transfer NFT back (75ms)
✓ Should refuse to process the transcation (auction stopped)
✓ Should restart the sale (91ms)
✓ Should process the transaction (185ms)
    Case 2 - wrong currency type
✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (136ms)
    Group ERC1155 | FakeWBTC
      Case 1
✓ Should refuse to start sale (NFT not approved) (57ms)
✓ Should start the sale (163ms)
✓ Should update the price (102ms)
✓ Should stop the sale and transfer NFT back (75ms)
```

- ✓ Should refuse to process the transcation (auction stopped) (42ms)
- ✓ Should restart the sale (143ms)
- ✓ Should process the transaction (248ms)

  Case 2 - wrong currency type
- ✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (52ms)

Group ERC1155 | ETH native

  Case 1
- ✓ Should refuse to start sale (NFT not approved) (59ms)
- ✓ Should start the sale (122ms)
- ✓ Should update the price (52ms)
- ✓ Should stop the sale and transfer NFT back (68ms)
- ✓ Should refuse to process the transcation (auction stopped)
- ✓ Should restart the sale (95ms)
- ✓ Should process the transaction (170ms)

  Case 2 - wrong currency type
- ✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (117ms)

Standard Auction

Group ERC721 | FakeWBTC

  Case 1 - minimal price not reached
- ✓ Should refuse to start auction (NFT not approved) (79ms)
- ✓ Should start the sale (114ms)
- ✓ Should place the bid (152ms)
- ✓ Should refuse to place the bid (offer too low)
- ✓ Should refuse to finalize the transaction (not finished yet) (46ms)
- ✓ Should refuse to finalize the transaction (not entitled to finish)
- ✓ Should refuse finish the transaction (minimal price not reached) (43ms)
- ✓ Should withdraw funds and NFT (131ms)

  Case 2 - no offers
- ✓ Should start the sale (124ms)
- ✓ Should refuse to finish the auction (no offers/minimal price not reached)
- ✓ Should withdraw the NFT (no offers) (67ms)

  Case 3 - standard flow
- ✓ Should update the config (auction not started yet)
- ✓ Should start the sale (99ms)
- ✓ Should refuse to update auction config (auction is ongoing)
- ✓ Should place the bid (146ms)
- ✓ Should place the bid and refund previous winner (208ms)
- ✓ Winner should be able to finish the auction (284ms)

  Case 4 - buy now enabled
- ✓ Should start the sale (124ms)
- ✓ Should refuse to return buyNow price (minimal price not reached)
- ✓ Should refuse to buyNow (minimal price not reached)

- ✓ Should bid above minimal price (120ms)
- ✓ Should return calculated buyNow price (58ms)
- ✓ Should process buyNow transcation (121ms)
- ✓ Should revert bid attempt (134ms)
- ✓ Should revert finish attempt
- ✓ Should revert withdraw attempt (39ms)

  Case 5 - buy now failed
- ✓ Should revert buyNow attempt (insufficient funds) (47ms)
- ✓ Bidding should still be available (161ms)

  Case 6 - auction cancelled
- ✓ Should cancel the auction and refund current winner (83ms)
- ✓ Should reset currentWinner and currentOffer when the auction is restarted (89ms)

  Case 7 - wrong currency
- ✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (45ms)

Group ERC721 | ETH native

  Case 1 - minimal price not reached
- ✓ Should refuse to start auction (NFT not approved) (60ms)
- ✓ Should start the sale (144ms)
- ✓ Should place the bid (68ms)
- ✓ Should refuse to place the bid (offer too low) (54ms)
- ✓ Should refuse to finalize the transaction (not finished yet)
- ✓ Should refuse to finalize the transaction (not entitled to finish)
- ✓ Should refuse finish the transaction (minimal price not reached)
- ✓ Should withdraw funds and NFT (111ms)

  Case 2 - no offers
- ✓ Should start the sale (292ms)
- ✓ Should refuse to finish the auction (no offers/minimal price not reached) (56ms)
- ✓ Should withdraw the NFT (no offers) (86ms)

  Case 3 - standard flow
- ✓ Should update the config (auction not started yet)
- ✓ Should start the sale (138ms)
- ✓ Should refuse to update auction config (auction is ongoing)
- ✓ Should place the bid (100ms)
- ✓ Should place the bid and refund previous winner (171ms)
- ✓ Winner should be able to finish the auction (141ms)

  Case 4 - buy now enabled
- ✓ Should start the sale (116ms)
- ✓ Should refuse to return buyNow price (minimal price not reached)
- ✓ Should refuse to buyNow (minimal price not reached)
- ✓ Should bid above minimal price (83ms)
- ✓ Should return calculated buyNow price (46ms)

- ✓ Should process buyNow transcation (114ms)
- ✓ Should revert bid attempt (51ms)
- ✓ Should revert finish attempt
- ✓ Should revert withdraw attempt

  Case 5 - buy now failed
- ✓ Should revert buyNow attempt (insufficient funds)
- ✓ Bidding should still be available (80ms)

  Case 6 - auction cancelled
- ✓ Should cancel the auction and refund current winner (58ms)
- ✓ Should reset currentWinner and currentOffer when the auction is restarted (110ms)

  Case 7 - wrong currency
- ✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (94ms)

  Group ERC1155 | FakeWBTC

  Case 1 - minimal price not reached
- ✓ Should refuse to start auction (NFT not approved)
- ✓ Should start the sale (94ms)
- ✓ Should place the bid (152ms)
- ✓ Should refuse to place the bid (offer too low)
- ✓ Should refuse to finalize the transaction (not finished yet) (49ms)
- ✓ Should refuse to finalize the transaction (not entitled to finish) (39ms)
- ✓ Should refuse finish the transaction (minimal price not reached)
- ✓ Should withdraw funds and NFT (129ms)

  Case 2 - no offers
- ✓ Should start the sale (154ms)
- ✓ Should refuse to finish the auction (no offers/minimal price not reached) (38ms)
- ✓ Should withdraw the NFT (no offers) (52ms)

  Case 3 - standard flow
- ✓ Should update the config (auction not started yet) (47ms)
- ✓ Should start the sale (133ms)
- ✓ Should refuse to update auction config (auction is ongoing)
- ✓ Should place the bid (136ms)
- ✓ Should place the bid and refund previous winner (202ms)
- ✓ Winner should be able to finish the auction (287ms)

  Case 4 - buy now enabled
- ✓ Should start the sale (119ms)
- ✓ Should refuse to return buyNow price (minimal price not reached)
- ✓ Should refuse to buyNow (minimal price not reached)
- ✓ Should bid above minimal price (94ms)
- ✓ Should return calculated buyNow price (68ms)
- ✓ Should process buyNow transcation (164ms)

- ✓ Should revert bid attempt (98ms)
- ✓ Should revert finish attempt
- ✓ Should revert withdraw attempt

  Case 5 - buy now failed
- ✓ Should revert buyNow attempt (insufficient funds) (123ms)
- ✓ Bidding should still be available (1160ms)

  Case 6 - auction cancelled
- ✓ Should cancel the auction and refund current winner (63ms)
- ✓ Should reset currentWinner and currentOffer when the auction is restarted (94ms)

  Case 7 - wrong currency
- ✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (55ms)

Group ERC1155 | ETH native

Case 1 - minimal price not reached
- ✓ Should refuse to start auction (NFT not approved) (64ms)
- ✓ Should start the sale (163ms)
- ✓ Should place the bid (84ms)
- ✓ Should refuse to place the bid (offer too low) (42ms)
- ✓ Should refuse to finalize the transaction (not finished yet) (39ms)
- ✓ Should refuse to finalize the transaction (not entitled to finish) (42ms)
- ✓ Should refuse finish the transaction (minimal price not reached)
- ✓ Should withdraw funds and NFT (110ms)

  Case 2 - no offers
- ✓ Should start the sale (99ms)
- ✓ Should refuse to finish the auction (no offers/minimal price not reached) (38ms)
- ✓ Should withdraw the NFT (no offers) (49ms)

  Case 3 - standard flow
- ✓ Should update the config (auction not started yet)
- ✓ Should start the sale (118ms)
- ✓ Should refuse to update auction config (auction is ongoing)
- ✓ Should place the bid (97ms)
- ✓ Should place the bid and refund previous winner (150ms)
- ✓ Winner should be able to finish the auction (217ms)

  Case 4 - buy now enabled
- ✓ Should start the sale (146ms)
- ✓ Should refuse to return buyNow price (minimal price not reached)
- ✓ Should refuse to buyNow (minimal price not reached)
- ✓ Should bid above minimal price (98ms)
- ✓ Should return calculated buyNow price (57ms)
- ✓ Should process buyNow transcation (71ms)

✓ Should revert bid attempt
✓ Should revert finish attempt
✓ Should revert withdraw attempt
Case 5 - buy now failed
✓ Should revert buyNow attempt (insufficient funds)
✓ Bidding should still be available (93ms)
Case 6 - auction cancelled
✓ Should cancel the auction and refund current winner (70ms)
✓ Should reset currentWinner and currentOffer when the auction is restarted (119ms)
Case 7 - wrong currency
✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (104ms)
Dutch Auction
Group ERC721 | FakeWBTC
Case 0 - auction preconditions
✓ Should revert deploy if price will go below zero (116ms)
✓ Should revert deploy if provided non-contract address (66ms)
✓ Should revert deploy if provided contract is not IERC165 (91ms)
Case 1 - standard flow
✓ Should revert currentPrice if auction not started
✓ Should refuse to start auction (nft transfer not approved) (46ms)
✓ Should refuse to start auction (only owner can start)
✓ Should update the config (auction not started yet) (42ms)
✓ Should start auction and transfer NFT (180ms)
✓ Should refuse to re-start the auction
✓ Should refuse to update auction config (auction is ongoing)
✓ Should read updated currentPrice
✓ Should process the payment and NFT transfer (317ms)
✓ Should revert stopAuction if item is sold
Case 2 - auction ends without sell
✓ Should return updated price after 5 iterations
✓ Should revert currentPrice if auction is ended
✓ Should revert transaction if auction is ended
✓ Should be able to stop auction and withdraw NFT (62ms)
Case 3 - item is already sold
✓ Should revert bid if item already sold (251ms)
✓ Should revert currentPrice call if item already sold (45ms)
✓ Should revert startAuction if item already sold
Case 4 - royalties transfer
✓ Should complete transaction and transfer the royalties (274ms)
Case 5 - royalties & platform fee
✓ Should complete transaction and transfer the royalties & platform fee (318ms)

Case 6 - auction cancelled
✓ Should cancel the auction (52ms)
✓ Should revert buy if auction is cancelled
Case 7 - wrong currency type
✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (72ms)
Group ERC721 | ETH native
Case 0 - auction preconditions
✓ Should revert deploy if price will go below zero (75ms)
✓ Should revert deploy if provided non-contract address (78ms)
✓ Should revert deploy if provided contract is not IERC165 (70ms)
Case 1 - standard flow
✓ Should revert currentPrice if auction not started
✓ Should refuse to start auction (nft transfer not approved) (41ms)
✓ Should refuse to start auction (only owner can start)
✓ Should update the config (auction not started yet) (45ms)
✓ Should start auction and transfer NFT (116ms)
✓ Should refuse to re-start the auction
✓ Should refuse to update auction config (auction is ongoing)
✓ Should read updated currentPrice
✓ Should process the payment and NFT transfer (149ms)
✓ Should revert stopAuction if item is sold
Case 2 - auction ends without sell
✓ Should return updated price after 5 iterations
✓ Should revert currentPrice if auction is ended
✓ Should revert transaction if auction is ended
✓ Should be able to stop auction and withdraw NFT (50ms)
Case 3 - item is already sold
✓ Should revert bid if item already sold (101ms)
✓ Should revert currentPrice call if item already sold
✓ Should revert startAuction if item already sold
Case 4 - royalties transfer
✓ Should complete transaction and transfer the royalties (157ms)
Case 5 - royalties & platform fee
✓ Should complete transaction and transfer the royalties & platform fee (139ms)
Case 6 - auction is cancelled
✓ Should cancel the auction
✓ Should revert buy if auction is cancelled
Case 7 - wrong currency type
✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (165ms)

Group ERC1155 | FakeWBTC
Case 0 - auction preconditions
✓ Should revert deploy if price will go below zero (58ms)
✓ Should revert deploy if provided non-contract address
✓ Should revert deploy if provided contract is not IERC165 (56ms)
Case 1 - standard flow
✓ Should revert currentPrice if auction not started
✓ Should refuse to start auction (nft transfer not approved)
✓ Should refuse to start auction (only owner can start)
✓ Should update the config (auction not started yet)
✓ Should start auction and transfer NFT (82ms)
✓ Should refuse to re-start the auction
✓ Should refuse to update auction config (auction is ongoing)
✓ Should read updated currentPrice
✓ Should process the payment and NFT transfer (215ms)
✓ Should revert stopAuction if item is sold
Case 2 - auction ends without sell
✓ Should return updated price after 5 iterations
✓ Should revert currentPrice if auction is ended
✓ Should revert transaction if auction is ended
✓ Should be able to stop auction and withdraw NFT (45ms)
Case 3 - item is already sold
✓ Should revert bid if item already sold (134ms)
✓ Should revert currentPrice call if item already sold
✓ Should revert startAuction if item already sold
Case 4 - royalties transfer
✓ Should complete transaction and transfer the royalties (238ms)
Case 5 - royalties & platform fee
✓ Should complete transaction and transfer the royalties & platform fee (201ms)
Case 6 - auction is cancelled
✓ Should cancel the auction (50ms)
✓ Should revert buy if auction is cancelled
Case 7 - wrong currency type
✓ Should revert bid with incorrect currency (valid ERC20, invalid: NATIVE) (51ms)
Group ERC1155 | ETH native
Case 0 - auction preconditions
✓ Should revert deploy if price will go below zero (44ms)
✓ Should revert deploy if provided non-contract address (41ms)
✓ Should revert deploy if provided contract is not IERC165 (56ms)

Case 1 - standard flow
✓ Should revert currentPrice if auction not started
✓ Should refuse to start auction (nft transfer not approved)
✓ Should refuse to start auction (only owner can start)
✓ Should update the config (auction not started yet)
✓ Should start auction and transfer NFT (81ms)
✓ Should refuse to re-start the auction
✓ Should refuse to update auction config (auction is ongoing)
✓ Should read updated currentPrice
✓ Should process the payment and NFT transfer (136ms)
✓ Should revert stopAuction if item is sold (38ms)
Case 2 - auction ends without sell
✓ Should return updated price after 5 iterations
✓ Should revert currentPrice if auction is ended
✓ Should revert transaction if auction is ended
✓ Should be able to stop auction and withdraw NFT (132ms)
Case 3 - item is already sold
✓ Should revert bid if item already sold (119ms)
✓ Should revert currentPrice call if item already sold
✓ Should revert startAuction if item already sold
Case 4 - royalties transfer
✓ Should complete transaction and transfer the royalties (217ms)
Case 5 - royalties & platform fee
✓ Should complete transaction and transfer the royalties & platform fee (131ms)
Case 6 - auction is cancelled
✓ Should cancel the auction (45ms)
✓ Should revert buy if auction is cancelled
Case 7 - wrong currency type
✓ Should revert bid with incorrect currency (valid NATIVE, invalid: ERC20) (125ms)
COLLECTIONS
ERC721
General
✓ Should implement the IERC721 interface
✓ Should implement the IERC721Metadata interface
✓ Name and symbol should match the constructor params
✓ Should mint new NFT of id 1 to owner (62ms)
Transfers
✓ Should transfer NFT of id 1 to addr1 (64ms)
✓ Should fail transfer of non-existing token
✓ Should fail if address is not an NFT owner

URI modification
✓ Should fail to update URI if not contract owner
✓ Should update non-freezed URI
✓ Should fail to modify freezed URI (62ms)
Royalites
✓ Should implement the IERC2981 interface
✓ Should properly calculate the royalty
ERC1155
General
✓ Should implement the IERC165 interface
✓ Name and symbol should match the constructor params (58ms)
✓ Should mint one new NFT of id 1 to owner (55ms)
✓ Should mint ten new NFTs of id 2 to addr1 (45ms)
Transfers
✓ Should transfer NFT of id 1 to addr1 (68ms)
✓ Should fail to transfer 12 NFTs of id 2 from addr1 to addr2
URI modification
✓ Should fail to update URI if not contract owner (45ms)
✓ Should update non-freezed URI (46ms)
✓ Should fail to modify freezed URI (62ms)
Royalites
✓ Should implement the IERC2981 interface
✓ Should properly calculate the royalty (42ms)
FACTORIES
CollectionFactory
✓ Should create ERC721 collection
✓ Should create ERC1155 collection (55ms)
AuctionFactory - preconditions
Maintainers
✓ Should refuse to add maintainer (not a maintainer)
✓ Should refuse to remove maintainer (not a maintainer)
✓ Should add new maintainer
✓ Should remove maintainer (67ms)
Updating parameters
✓ Should refuse to update currency contract (not a maintainer) (40ms)
✓ Should refuse to update fee config (not a maintainer) (54ms)
✓ Should update currency (61ms)
✓ Should update feeConfig (59ms)

AuctionFactory - deployment
  Group ERC721
    Simple Sale
  ✓ Should deploy simple sale with safeTransfer (87ms)
  ✓ Should have proper config (101ms)
  ✓ Sender should be auction owner
  ✓ Should not change auction config if factory config is changed (48ms)
    Dutch bid
  ✓ Should deploy dutch auction (98ms)
  ✓ Should have proper config (44ms)
  ✓ Sender should be auction owner (51ms)
    Standard bid
  ✓ Should deploy standard auction (126ms)
  ✓ Should have proper config (62ms)
  ✓ Sender should be auction owner
  Group ERC1155
    Simple Sale
  ✓ Should deploy simple sale with safeTransfer (104ms)
  ✓ Should have proper config (55ms)
  ✓ Sender should be auction owner
  ✓ Should not change auction config if factory config is changed (45ms)
    Dutch bid
  ✓ Should deploy dutch auction (97ms)
  ✓ Should have proper config (55ms)
  ✓ Sender should be auction owner
    Standard bid
  ✓ Should deploy standard auction (82ms)
  ✓ Should have proper config (75ms)
  ✓ Sender should be auction owner
VAULT
  WBTC Vault
    Set 0 - preconditions
  ✓ Should refuse to deploy Vault (no owners) (49ms)
  ✓ Should refuse to deploy Vault (owner doubled) (39ms)
  ✓ Should refuse to deploy Vault (address zero as owner)
  ✓ Should deploy the Vault (76ms)
  ✓ Should return array of owners
    Set 1 - submitting
  ✓ Should refuse to submit transaction (zero address recipient)
  ✓ Should refuse to submit transaction (zero value) (39ms)
  ✓ Should refuse to submit transaction (not owner) (50ms)

✓ Should submit transaction (48ms)
✓ Should read submitted transaction details (38ms)
Set 2 - confirmation and execution
✓ Should refuse to execute transaction (tx does not exist)
✓ Should refuse to execute transaction (not enough confirmations) (49ms)
✓ Should refuse to execute transaction (not owner) (45ms)
✓ Should refuse to confirm transaction (not owner) (40ms)
✓ Should refuse to confirm transaction (tx does not exist)
✓ Owners should confirm transaction (67ms)
✓ Should refuse to reconfirm transaction (already confirmed) (50ms)
✓ Should fail to execute transaction (balance too low) (54ms)
✓ Last confirmation should execute transaction (160ms)
✓ Should fail to re-execute transaction (already executed)
Set 3 - revoking confirmation
✓ Should revoke confirmation (85ms)
mintfee
Mint fee - ERC721
✓ Factory maintainer should update fee details (53ms)
✓ Should transfer the mint fee (45ms)
✓ Should revert minting without sending fee
Mint fee - ERC1155
✓ Factory maintainer should update fee details (98ms)
✓ Should transfer the mint fee (53ms)
✓ Should revert minting without sending fee

336 passing (51s)

FILE	% STMTS	% BRANCH	% FUNCS
AuctionFactory.sol	100	64.29	100
CollectionFactory.sol	100	50	100
DutchBid.sol	100	100	100
NFTCollectionERC1155.sol	100	100	100
NFTCollectionERC721.sol	100	66.67	100
PlatformAuction.sol	96.61	93.75	100
SimpleSale.sol	100	100	100
StandardBid.sol	96	84.62	100
Vault.sol	97.37	95.83	91.67
FeePayer.sol	85.71	83.33	100
Maintainable.sol	100	100	100
All files	**97.4**	**86.67**	**97.8**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting OneMind in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as the review of the OneMind contract requirements for details about issuance amounts and how the system handles these.

Contract: AuctionFactory

Maintainer operations

- ✓ Shouldn't add maintainer by someone) (51ms)
- ✓ Shouldn't remove maintainer by someone
- ✓ Should add maintainer (61ms)
- ✓ Shouldn't remove yourself from maintainers (65ms)
- ✓ Should remove maintainer (136ms)

Updating parameters

- ✓ Should refuse to update currency contract by not a maintainer (60ms)
- ✓ Should refuse to update fee config by not a maintainer (56ms)
- ✓ Should update currency (59ms)
- ✓ Should update feeConfig (52ms)
- ✓ Shouldn't update feeConfig with treasury zero address
- ✓ Shouldn't deploy auction factory with treasury zero address (90ms)

Create Auction

- ✓ Should create auction simple sale with erc1155 (253ms)
- ✓ Should create dutch auction with erc1155 (219ms)
- ✓ Should create standard auction with erc1155 (212ms)
- ✓ Should create auction simple sale with erc721 (252ms)
- ✓ Should create dutch auction with erc721 (186ms)
- ✓ Should create standard auction with erc721 (215ms)

Contract: CollectionFactory

- ✓ Should create ERC721 collection (46ms)
- ✓ Should create ERC1155 collection (76ms)
- ✓ Should update fee details by factory maintainer (38ms)
- ✓ Shouldn't update fee details if treasury is zero address
- ✓ Should update fee details only by maintainer

Contract: DutchBid

Dutch bid with erc721

- ✓ Should revert deploy if price will go below zero (71ms)
- ✓ Should revert deploy if provided non-contract address (62ms)

- ✓ Should revert deploy if provided contract is not IERC165 (61ms)
- ✓ Should start the dutch auction (112ms)
- ✓ Should revert currentPrice if auction not started
- ✓ Shouldn't start auction if nft transfer not approved (53ms)
- ✓ Shouldn't start auction by someone
- ✓ Shouldn't update config with invalid params
- ✓ Should update the config if auction not started yet (43ms)
- ✓ Shouldn't re-start the auction (118ms)
- ✓ Shouldn't update auction config if auction is ongoing (104ms)
- ✓ Shouldn't stop auction if item is sold (263ms)
- ✓ Should bid (236ms)

Dutch bid with erc1155

- ✓ Should revert deploy if price will go below zero (56ms)
- ✓ Should revert deploy if provided non-contract address (74ms)
- ✓ Should revert deploy if provided contract is not IERC165 (62ms)
- ✓ Should start the dutch auction (215ms)
- ✓ Shouldn't get current price if auction not started
- ✓ Shouldn't start auction if nft transfer not approved (78ms)
- ✓ Shouldn't start auction by someone
- ✓ Should update the config if auction not started yet (43ms)
- ✓ Shouldn't restart the auction (117ms)
- ✓ Shouldn't update auction config if auction is ongoing (139ms)
- ✓ Shouldn't stop auction if item is sold (208ms)

Contract: SimpleSale

- ✓ Should finalize auction if currency zero address (221ms)
- ✓ Shouldn't deploy contract with invalid NFT type (117ms)
- ✓ Shouldn't bid if msg.value != price and currency is zero address (272ms)

Simple sale with erc721

- ✓ Simple sale with erc721
- ✓ Shouldn't start sale if NFT not approved (74ms)
- ✓ Should update the price (60ms)
- ✓ Should stop the sale and transfer NFT back (191ms)
- ✓ Shouldn't bid when auction stopped
- ✓ Should bid (401ms)

Simple sale with erc1155

- ✓ Should start the sale (193ms)
- ✓ Shouldn't start sale if NFT not approved (78ms)
- ✓ Should update the price (66ms)
- ✓ Should stop the sale and transfer NFT back (204ms)
- ✓ Shouldn't bid when auction stopped
- ✓ Should bid (270ms)
- ✓ Shouldn't finalize auction if nft transfer failed (374ms)

Contract: StandardBid

Deploy with invalid params

- ✓ Shouldn't deploy standard bid with buyNowPercent > 10000 (153ms)
- ✓ Shouldn't deploy standard bid with initial price > minimal price (181ms)
- ✓ Shouldn't deploy standard bid with auction duration = 0 (131ms)
- ✓ Shouldn't deploy standard bid with initial price = 0 (117ms)
- ✓ Shouldn't deploy standard bid with nominator > 10000 (97ms)

Standard bid with erc721

- ✓ Shouldn't start auction if NFT not approved (47ms)
- ✓ Should start standard auction (178ms)
- ✓ Should start standard auction if nft already stored on contract (161ms)
- ✓ Should start standard auction if nft already stored on contract (189ms)
- ✓ Should bid (231ms)
- ✓ Shouldn't bid if offer transfer failed (167ms)
- ✓ Shouldn't bid if offer too low (136ms)
- ✓ Shouldn't finalize the transaction if not finished yet (227ms)
- ✓ Shouldn't finalize the transaction if not entitled to finish (224ms)
- ✓ Shouldn't finish the transaction if minimal price not reached (207ms)
- ✓ Should update the config (auction not started yet)
- ✓ Shouldn't update the config if auction started (147ms)
- ✓ Should bid and refund previous winner (421ms)
- ✓ Should finish the auction by winner (478ms)
- ✓ Should withdraw funds and NFT (315ms)
- ✓ Shouldn't finish the auction if no offers/minimal price not reached (102ms)
- ✓ Shouldn't stop auction if item is sold (358ms)
- ✓ Shouldn't withdraw if auction is finished (302ms)
- ✓ Shouldn't bid if auction time ended (169ms)
- ✓ Should withdraw the NFT with no offers (123ms)

Standard bid with erc721 with buyNow setup

- ✓ Shouldn't return buyNow price if minimal price not reached (70ms)
- ✓ Shouldn't buyNow if minimal price not reached (97ms)
- ✓ Should return calculated buyNow price (211ms)
- ✓ Should buyNow (408ms)
- ✓ Should revert bid attempt (393ms)
- ✓ Should revert finish attempt (262ms)
- ✓ Should revert withdraw attempt (284ms)
- ✓ Shouldn't buyNow if insufficient funds
- ✓ Should cancel the auction and refund current winner (210ms)
- ✓ Should reset currentWinner and currentOffer when the auction is restarted (263ms)

Standard bid with erc1155

- ✓ Shouldn't start auction if NFT not approved (39ms)
- ✓ Should start standard auction (112ms)
- ✓ Should start standard auction if nft already stored on contract (145ms)

- ✓ Should bid (502ms)
- ✓ Shouldn't bid if offer too low (136ms)
- ✓ Should finalize auction and transfer nft to auction owner if nft transfer to winner failed (324ms)
- ✓ Shouldn't finalize the transaction if not finished yet (180ms)
- ✓ Shouldn't finalize the transaction if not entitled to finish (203ms)
- ✓ Shouldn't finish the transaction if minimal price not reached (169ms)
- ✓ Should update the config (auction not started yet)
- ✓ Shouldn't update the config if auction started (100ms)
- ✓ Should bid and refund previous winner (309ms)
- ✓ Should finish the auction by winner (332ms)
- ✓ Should withdraw funds and NFT (268ms)
- ✓ Shouldn't finish the auction if no offers/minimal price not reached (100ms)
- ✓ Shouldn't stop auction if item is sold (310ms)
- ✓ Shouldn't withdraw if auction is finished (266ms)
- ✓ Shouldn't bid if auction time ended (142ms)
- ✓ Should withdraw the NFT with no offers (119ms)

Standard bid with erc115 with buyNow setup

- ✓ Shouldn't return buyNow price if minimal price not reached (82ms)
- ✓ Shouldn't buyNow if minimal price not reached (81ms)
- ✓ Should return calculated buyNow price (183ms)
- ✓ Should buyNow (267ms)
- ✓ Should revert bid attempt (314ms)
- ✓ Should revert finish attempt (286ms)
- ✓ Should revert withdraw attempt (283ms)
- ✓ Shouldn't buyNow if insufficient funds
- ✓ Should cancel the auction and refund current winner (196ms)
- ✓ Should reset currentWinner and currentOffer when the auction is restarted (250ms)

Contract: NFTCollectionERC1155

- ✓ Admin should mint new NFT (89ms)
- ✓ Admin should not mint new NFT to Zero Address (69ms)
- ✓ Not admin should not mint new NFT
- ✓ Should get correct token royalty amount
- ✓ Should get correct token URI (40ms)
- ✓ Token URI if token is not exist is token base URI (50ms)
- ✓ Admin should update token URI (44ms)
- ✓ Not admin should not update token URI
- ✓ Admin should freeze existent token
- ✓ Not admin should not freeze token
- ✓ Admin should set new royalty for token
- ✓ Admin should not update token URI for freezed token (38ms)
- ✓ Admin should freeze token twice (test may falling)
- ✓ NFT contract should support ERC1155 interface

Contract: NFTCollectionERC721

- ✓ Admin should mint new NFT
- ✓ Admin should not mint new NFT to Zero Address
- ✓ Not admin should not mint new NFT
- ✓ Should get correct token royalty amount
- ✓ Should get correct token URI
- ✓ Should not get token URI if token is not exist
- ✓ Admin should update token URI
- ✓ Admin should not update token URI for nonexistent token
- ✓ Should cancel the auction and refund current winner (196ms)
- ✓ Admin should freeze existent token
- ✓ Admin should not freeze nonexistent token (70ms)
- ✓ Not admin should not freeze token
- ✓ Admin should set new royalty for token
- ✓ Admin should not set new royalty for nonexistent token
- ✓ Admin should not update token URI for freezed token (38ms)
- ✓ Admin should freeze token twice (test may falling)
- ✓ NFT contract should support ERC721 interface

Contract: Vault

Test constructor

- ✓ Correct deploy (138ms)
- ✓ Error if currency is AddressZero (105ms)
- ✓ Error if list of owners is empty (85ms)
- ✓ Error if one of proposed owners is zero address (92ms)
- ✓ Error if proposed owners are repeated (91ms)

Test functions

- ✓ Should get owners
- ✓ Should get transaction count
- ✓ Should get transaction
- ✓ submitTransaction() Owner should submit transaction
- ✓ .submitTransaction() Not owner should not submit transaction
- ✓ .submitTransaction() Owner should not submit transaction to AddressZero
- ✓ .submitTransaction() Owner should not submit transaction with value <= 0
- ✓ .confirmTransaction() Owner should not confirm not exist transaction
- ✓ .confirmTransaction() Owner should not confirm transaction twice
- ✓ .confirmTransaction() Transaction must executing if it has all confirmations (64ms)
- ✓ Transaction should not executing if it expired (59ms)
- ✓ .confirmTransaction() Owner should not confirm executed transaction (69ms)
- ✓ .executeTransaction() Owner should not execute transaction if it has not all confirmations
- ✓ .revokeConfirmation() Owner should revoke confirmation
- ✓ .revokeConfirmation() Owner should not revoke confirmation if he not confirmed it

Additional cases

- ✓ Transaction should not executing if one of confirmation was revoke (53ms)

178 passing (36s)

FILE	% STMTS	% BRANCH	% FUNCS
AuctionFactory.sol	100	78.57	100
CollectionFactory.sol	100	100	100
DutchBid.sol	100	100	100
NFTCollectionERC1155.sol	100	100	100
NFTCollectionERC721.sol	100	100	100
PlatformAuction.sol	98.18	96.43	100
SimpleSale.sol	100	100	100
StandardBid.sol	95.16	88.89	100
Vault.sol	100	96.43	100
FeePayer.sol	87.5	60	100
Maintainable.sol	100	100	100
PlatformAuctionUtils.sol	100	100	100
All files	**98.14**	**93.13**	**100**

We are grateful for the opportunity to work with the OneMind team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the OneMind team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo