Blaize.Security

March 22nd, 2023 / V. 1.0

BINARYX

BINARYX

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Binaryx contract's source code was taken from the repository provided by the Binaryx Protocol team.

SCORE **9.8**/10

The scope of the project includes Binaryx set of contracts:

contracts\v3
AccessManager.sol
AddressesProvider.sol
Asset.sol
AssetPriceOracle.sol
BNRXToken.sol
CommissionsDistributor.sol
CoreManager.sol
KycStore.sol
Oracle.sol
OracleFactory.sol
PropertyFactory.sol
RewardsDistributor.sol
UiProvider.sol
UsdtfToken.sol

Repository:

https://github.com/binaryx-protocol/binaryx_app

Branch: main

Initial commit:

- 9e31bfbaf39d49b15c8ebf6ed3e3bd58aa09e5cd

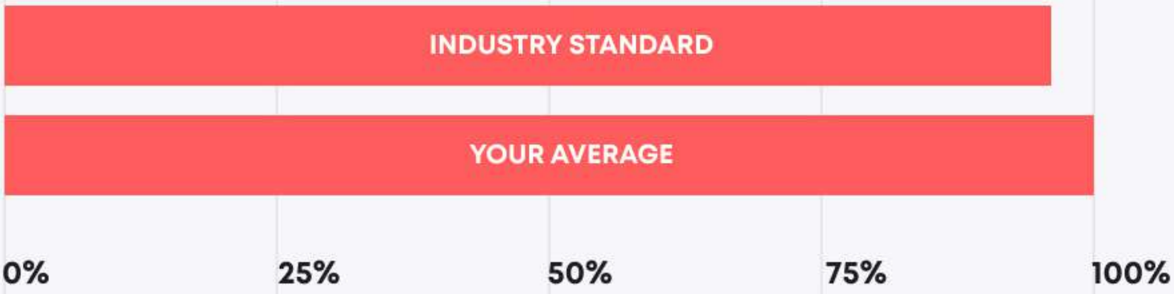
Final commit:

- 092fbce303350b5e95fa767e47127595b2442791

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the Binaryx protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Binaryx** smart contracts conducted between **February 17th, 2022** and **March 22nd, 2022.**

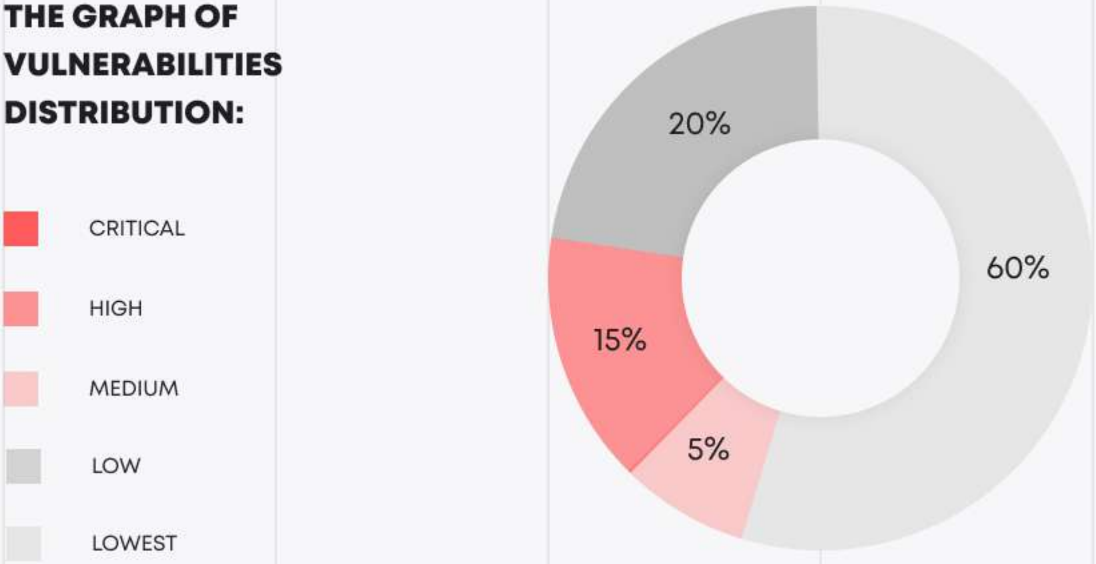
Testable code



The code is 100% testable, which corresponds to the industry standard of 95%.

The scope of the audit includes the unit test coverage, which is based on the smart contract code, documentation and requirements presented by the Binaryx team. The coverage is calculated based on the set of Hardhat framework tests and scripts from additional testing strategies. However, to ensure the security of the contract, the Blaize.Security team suggests that the Binaryx team launch a bug bounty program to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 16 problems were found. 14 issues were fixed or verified by the Binaryx team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	2	2
Medium	1	1
Low	5	5
Lowest	8	6

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

We checked the contract for the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets the best practices in the efficient use of gas, code readability.

Automated analysis:

Scanning contracts by several publicly available automated analysis tools such as Mythril, Solhint, Slither, and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. We checked smart contract logic and compared it with the one described in the documentation.

EXECUTIVE SUMMARY

The audited protocol is a marketplace of tokenized real estate that bridges the real estate market and the rapidly developing world of DeFi projects. The contracts are designed in such a way that they allow the administrator to have full control over the status of the asset, while the money of the user who participates in the purchase of the asset is completely under his control.

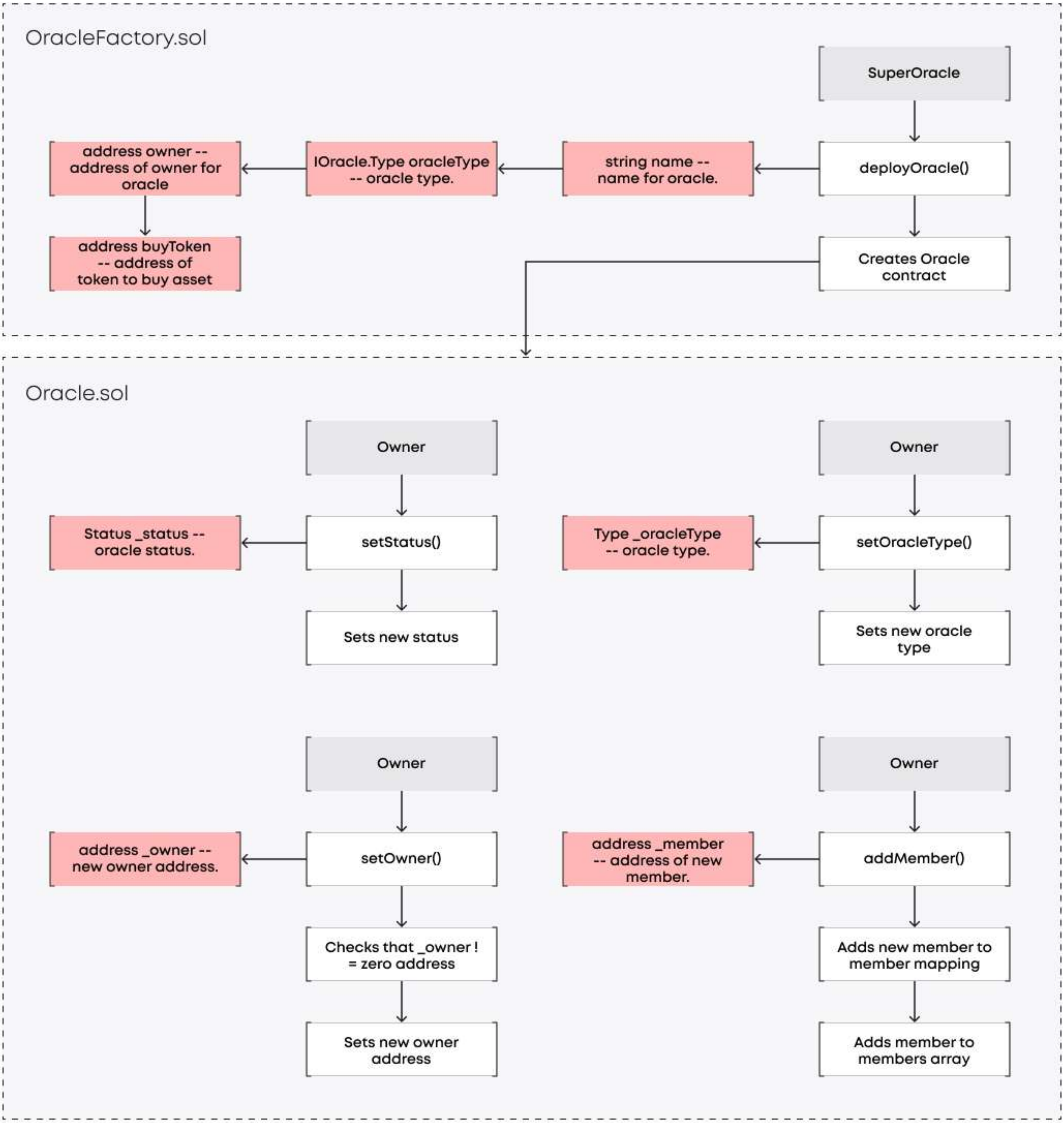
Contracts have all the possibilities for selling assets. All conditions of sale are transparent to the user. The role of the administrator is only to change the state of an asset. Likewise, the rules by which changes in the state of an asset occur are fixed in smart contracts.

No critical issues were found. Two high issues were associated with asset price changes down and the ability of the admin to change the state to any of the states without fixed conditions. These two issues have been fixed, as have others.
The overall security is high enough. Contracts' code has good readability and fulfills the necessary logic.

	RATING
Security	9.8
Gas usage and logic optimization	9.5
Code quality	10
Test coverage	10
Total	9.8

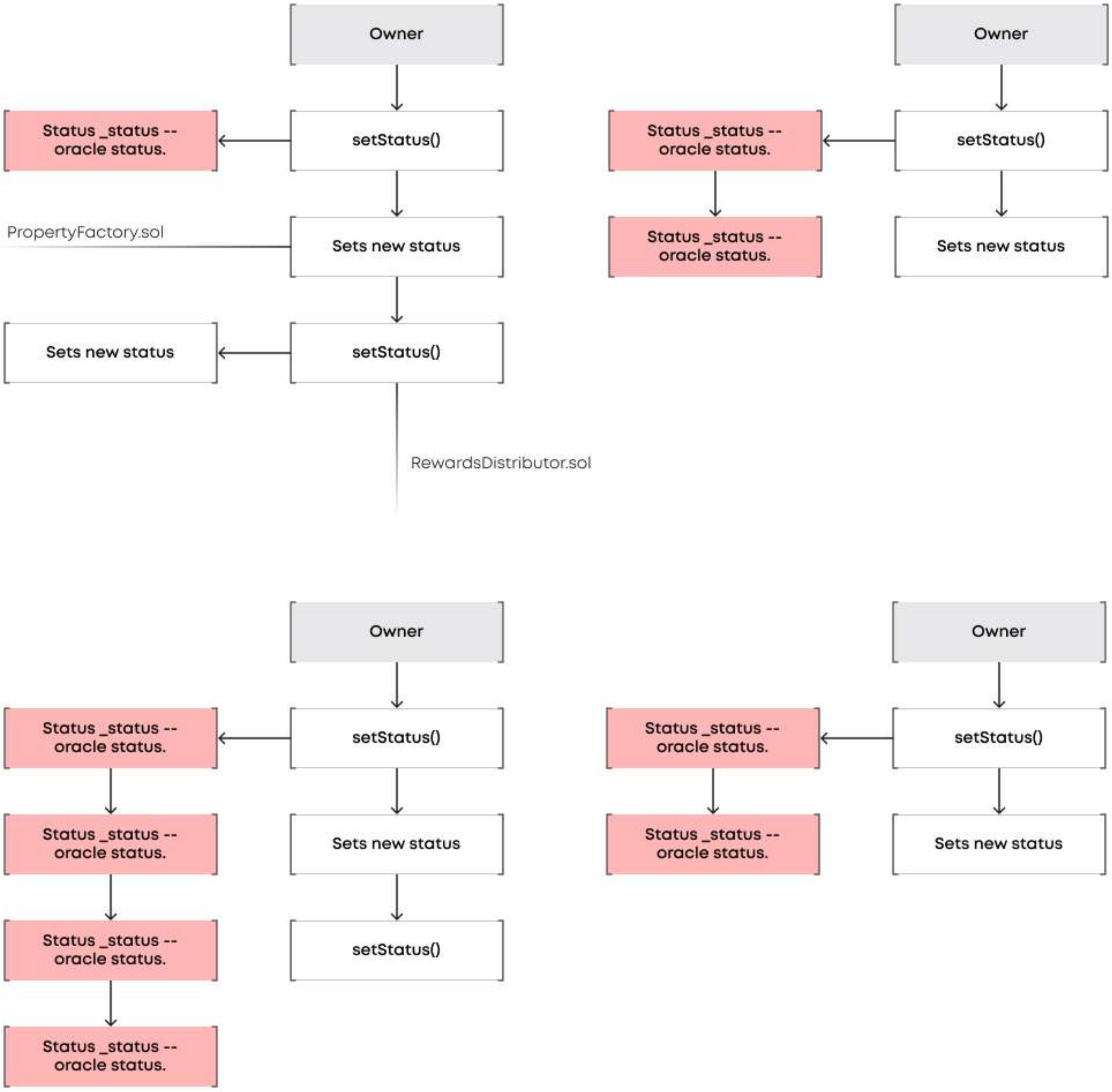
BINARYX

The OracleFactory contract serves as a central hub for managing oracles within the protocol. It provides a streamlined interface for administrators to list new oracles and get already deployed oracles.


OracleFactory.sol
SuperOracle
deployOracle()
string name -- name for oracle.
IOracle.Type oracleType -- oracle type.
address owner -- address of owner for oracle
address buyToken -- address of token to buy asset
Creates Oracle contract
Oracle.sol
Owner
setStatus()
Status _status -- oracle status.
Sets new status
Owner
setOracleType()
Type _oracleType -- oracle type.
Sets new oracle type
Owner
setOwner()
address _owner -- new owner address.
Checks that _owner ! = zero address
Sets new owner address
Owner
addMember()
address _member -- address of new member.
Adds new member to member mapping
Adds member to members array

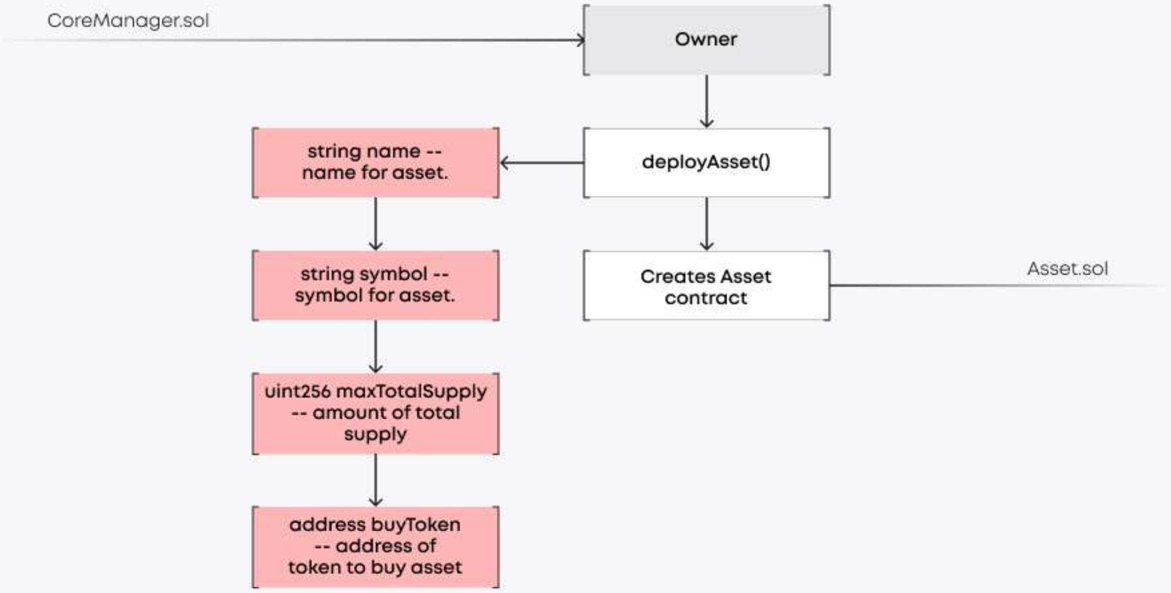
CoreManager.sol

The CoreManager contract serves as a central hub for managing assets within the protocol. It provides a streamlined interface for administrators to list new assets, update asset data and documents, and collect rental fees from users. This contract is a crucial component of the protocol's asset management system, and it helps to ensure that assets are managed efficiently and transparently.


Owner
setStatus()
Status _status -- oracle status.
Sets new status
PropertyFactory.sol
setStatus()
Sets new status
RewardsDistributor.sol
Owner
setStatus()
Status _status -- oracle status.
Status _status -- oracle status.
Sets new status
Owner
setStatus()
Status _status -- oracle status.
Status _status -- oracle status.
Status _status -- oracle status.
Status _status -- oracle status.
Sets new status
setStatus()
Owner
setStatus()
Status _status -- oracle status.
Status _status -- oracle status.
Sets new status

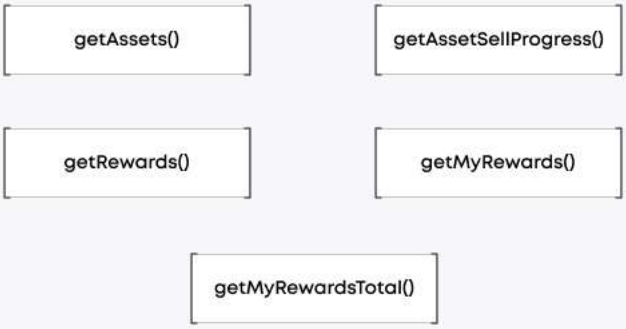
PropertyFactory.sol

The PropertyFactory contract serves as a central hub for managing assets within the protocol. It provides a streamlined interface for administrators to list new assets and get already deployed assets.


CoreManager.sol
Owner
deployAsset()
string name -- name for asset.
Creates Asset contract
Asset.sol
string symbol -- symbol for asset.
uint256 maxTotalSupply -- amount of total supply
address buyToken -- address of token to buy asset

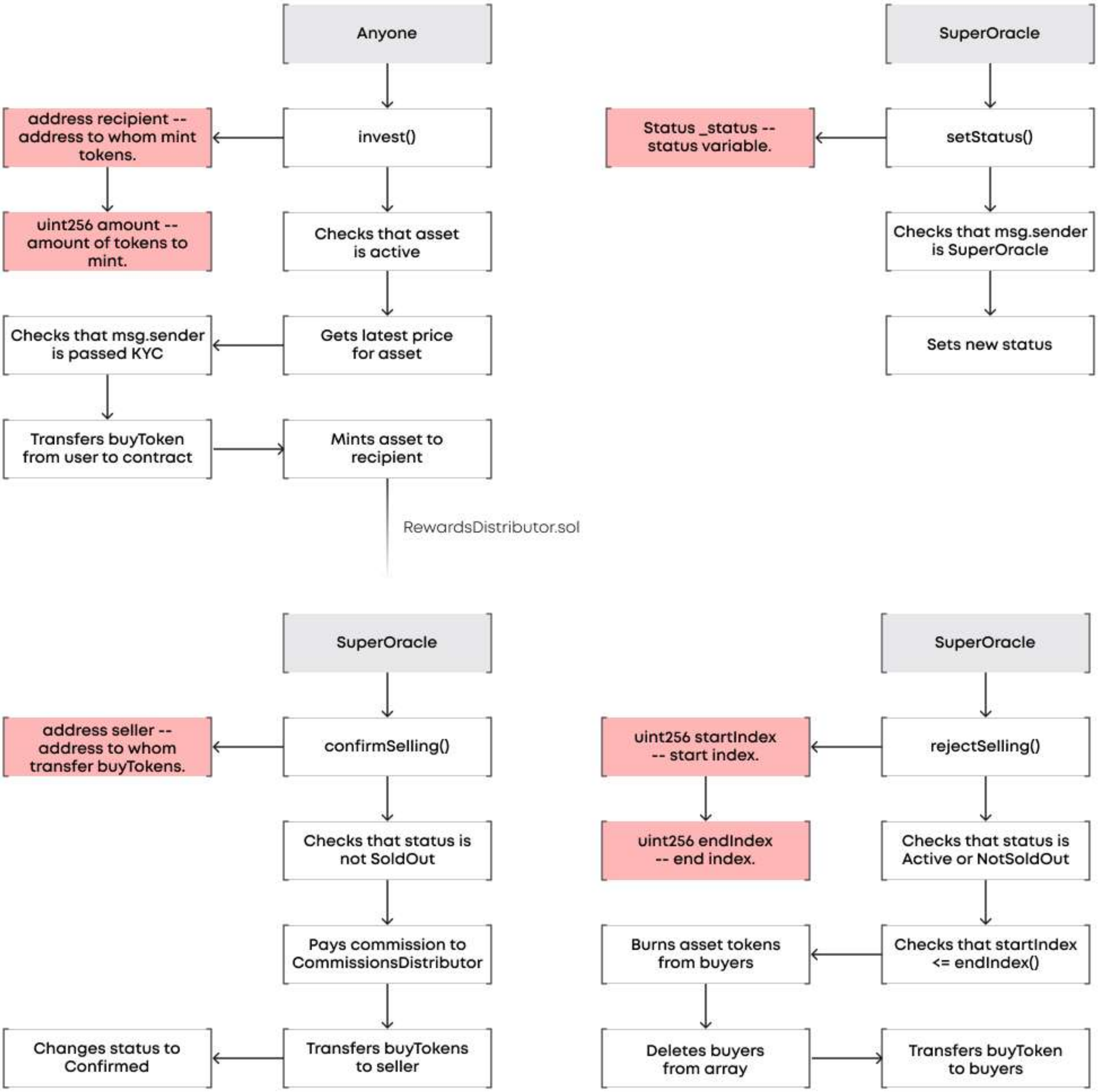
UiProvider.sol

The UiProvider contract optimizes data queries from the UI by aggregating calls to different contracts and collecting data that needs to be displayed on the UI. This reduces the number of calls that need to be made to the blockchain, which can improve performance and reduce the number of requests. The contract provides a simplified and efficient way to access data and enables a better user experience.



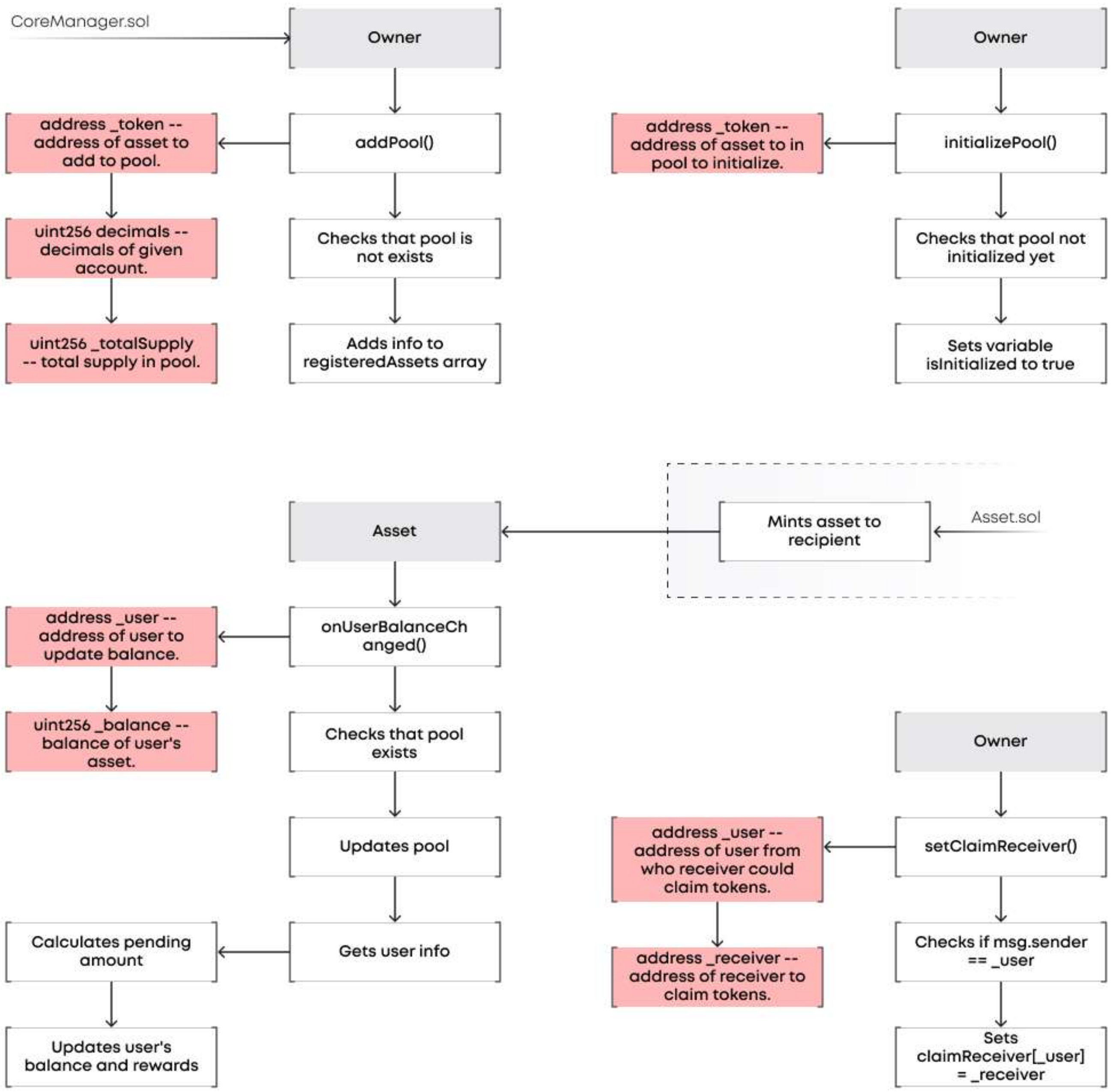
Asset.sol

The Asset contract is a foundational protocol component, providing the basic functionality for managing individual property tokens. It enables users to invest in assets by purchasing tokens and tracks the status of those tokens to ensure that investments are handled accurately. Additionally, it manages the storage of asset data and documents, as well as the asset update mechanism.


Anyone
invest()
address recipient -- address to whom mint tokens.
uint256 amount -- amount of tokens to mint.
Checks that asset is active
Gets latest price for asset
Checks that msg.sender is passed KYC
Transfers buyToken from user to contract
Mints asset to recipient
RewardsDistributor.sol
SuperOracle
setStatus()
Status _status -- status variable.
Checks that msg.sender is SuperOracle
Sets new status
SuperOracle
confirmSelling()
address seller -- address to whom transfer buyTokens.
Checks that status is not SoldOut
Pays commission to CommissionsDistributor
Transfers buyTokens to seller
Changes status to Confirmed
SuperOracle
rejectSelling()
uint256 startIndex -- start index.
uint256 endIndex -- end index.
Checks that status is Active or NotSoldOut
Checks that startIndex <= endIndex()
Burns asset tokens from buyers
Deletes buyers from array
Transfers buyToken to buyers

RewardsDistributor.sol

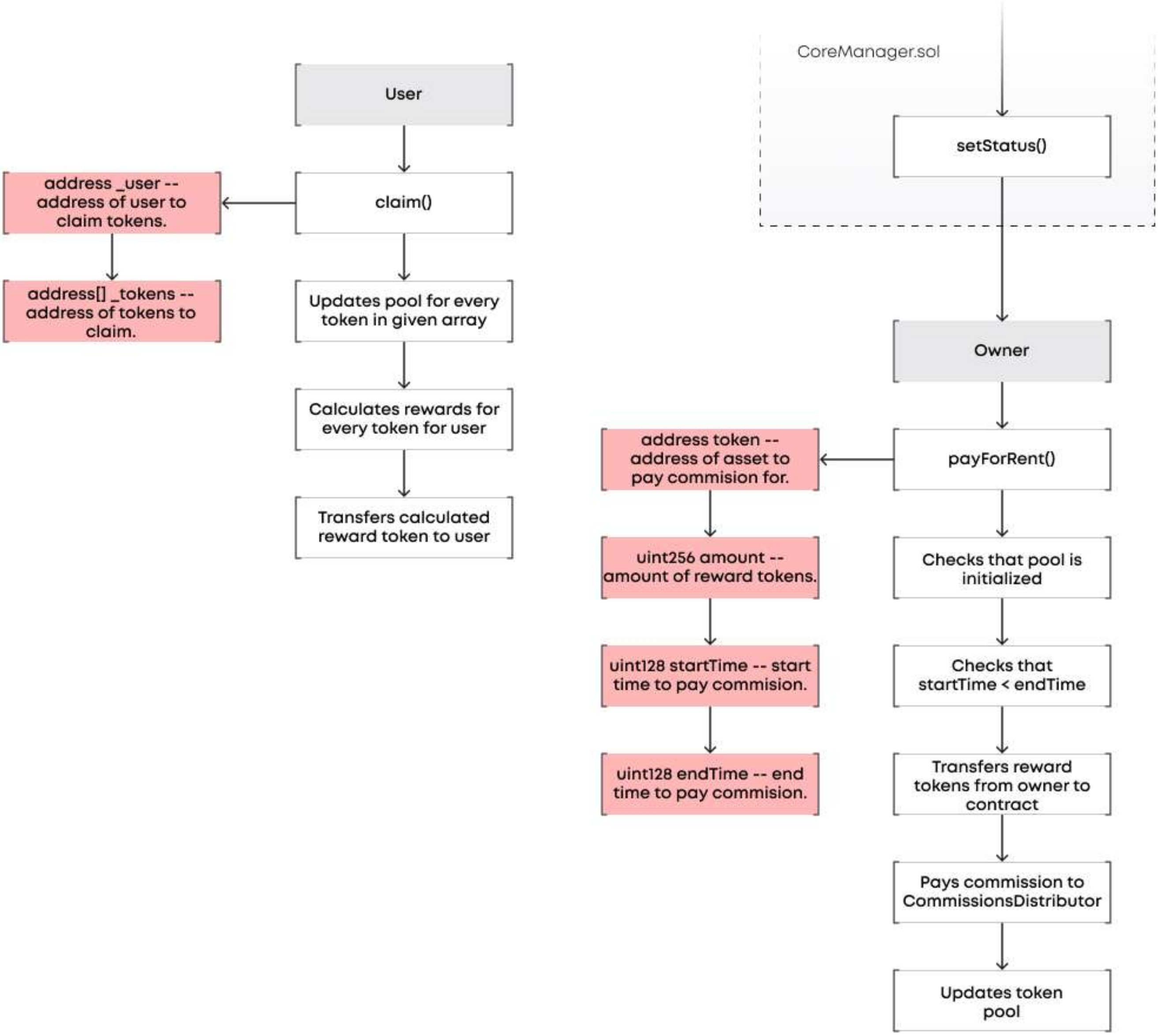
The RewardsDistributor contract is responsible for managing the distribution of rewards to investors based on their investments in the protocol. It tracks the share of investment in each token and uses this data to calculate and distribute rewards accordingly. Additionally, it allows administrators to pay for rent and distribute a portion of those fees as a commission to the protocol, with the remainder allocated for investment rewards.


CoreManager.sol
Owner
addPool()
address _token -- address of asset to add to pool.
uint256 decimals -- decimals of given account.
uint256 _totalSupply -- total supply in pool.
Checks that pool is not exists
Adds info to registeredAssets array
Owner
initializePool()
address _token -- address of asset to in pool to initialize.
Checks that pool not initialized yet
Sets variable isInitialized to true
Asset
Mints asset to recipient
Asset.sol
onUserBalanceChanged()
address _user -- address of user to update balance.
uint256 _balance -- balance of user's asset.
Checks that pool exists
Updates pool
Gets user info
Calculates pending amount
Updates user's balance and rewards
Owner
setClaimReceiver()
address _user -- address of user from who receiver could claim tokens.
address _receiver -- address of receiver to claim tokens.
Checks if msg.sender == _user
Sets claimReceiver[_user] = _receiver

BINARYX

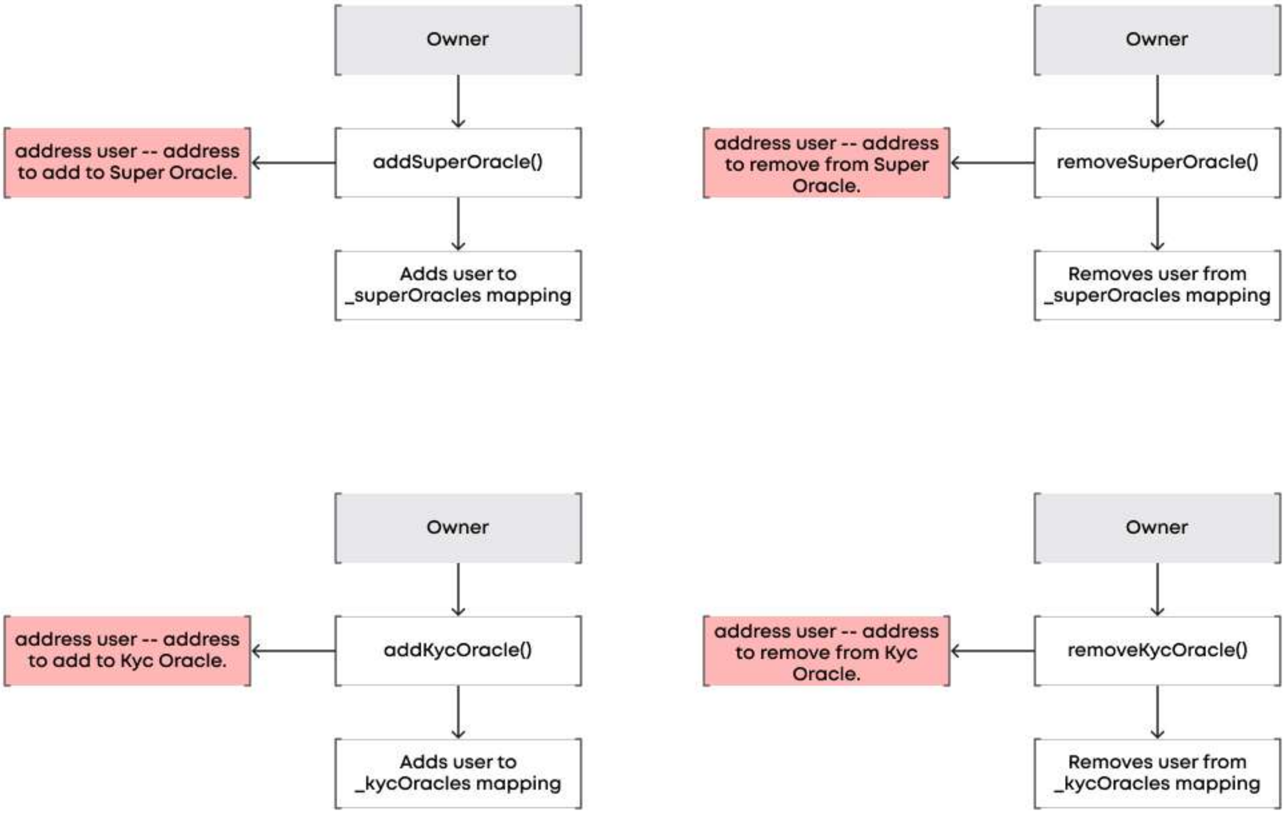
RewardsDistributor.sol

The RewardsDistributor contract is responsible for managing the distribution of rewards to investors based on their investments in the protocol. It tracks the share of investment in each token and uses this data to calculate and distribute rewards accordingly. Additionally, it allows administrators to pay for rent and distribute a portion of those fees as a commission to the protocol, with the remainder allocated for investment rewards.


User
claim()
address _user -- address of user to claim tokens.
address[] _tokens -- address of tokens to claim.
Updates pool for every token in given array
Calculates rewards for every token for user
Transfers calculated reward token to user
CoreManager.sol
setStatus()
Owner
payForRent()
address token -- address of asset to pay commision for.
uint256 amount -- amount of reward tokens.
uint128 startTime -- start time to pay commision.
uint128 endTime -- end time to pay commision.
Checks that pool is initialized
Checks that startTime < endTime
Transfers reward tokens from owner to contract
Pays commission to CommissionsDistributor
Updates token pool

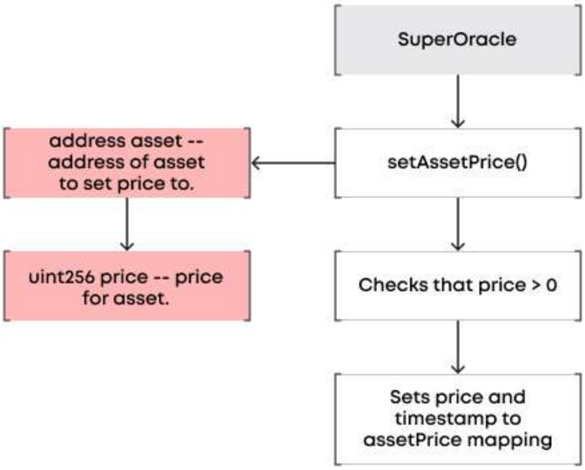
AccessManager.sol

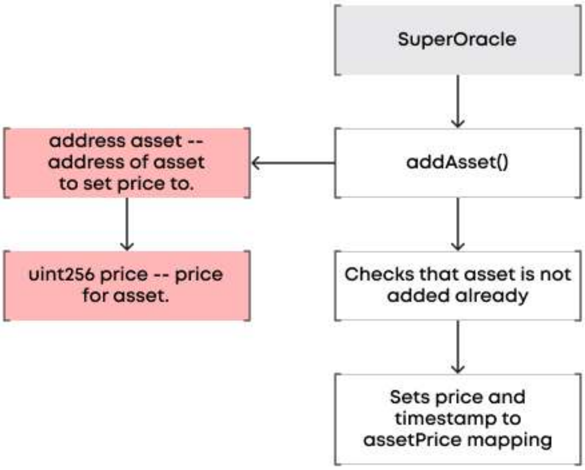
The AccessManager contract is responsible for managing access control to other smart contracts within the protocol. It allows administrators to define specific user roles and permissions, enabling fine-grained control over the protocol's operations. Additionally, it provides a mechanism for verifying access in other contracts, ensuring that only authorized users are able to perform certain actions.

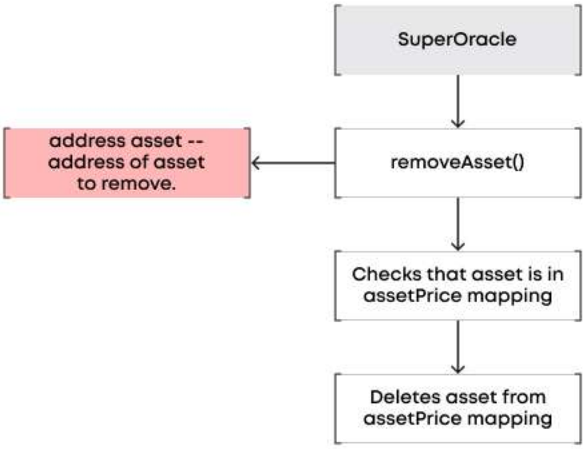

Owner
address user -- address to add to Super Oracle.
addSuperOracle()
Adds user to _superOracles mapping
Owner
address user -- address to remove from Super Oracle.
removeSuperOracle()
Removes user from _superOracles mapping
Owner
address user -- address to add to Kyc Oracle.
addKycOracle()
Adds user to _kycOracles mapping
Owner
address user -- address to remove from Kyc Oracle.
removeKycOracle()
Removes user from _kycOracles mapping

AssetPriceOracle.sol

The AssetPriceOracle contract is responsible for managing the prices of all real estate assets within the protocol. It enables administrators to add new assets and update prices as needed, ensuring the protocol's pricing data is accurate and up-to-date.

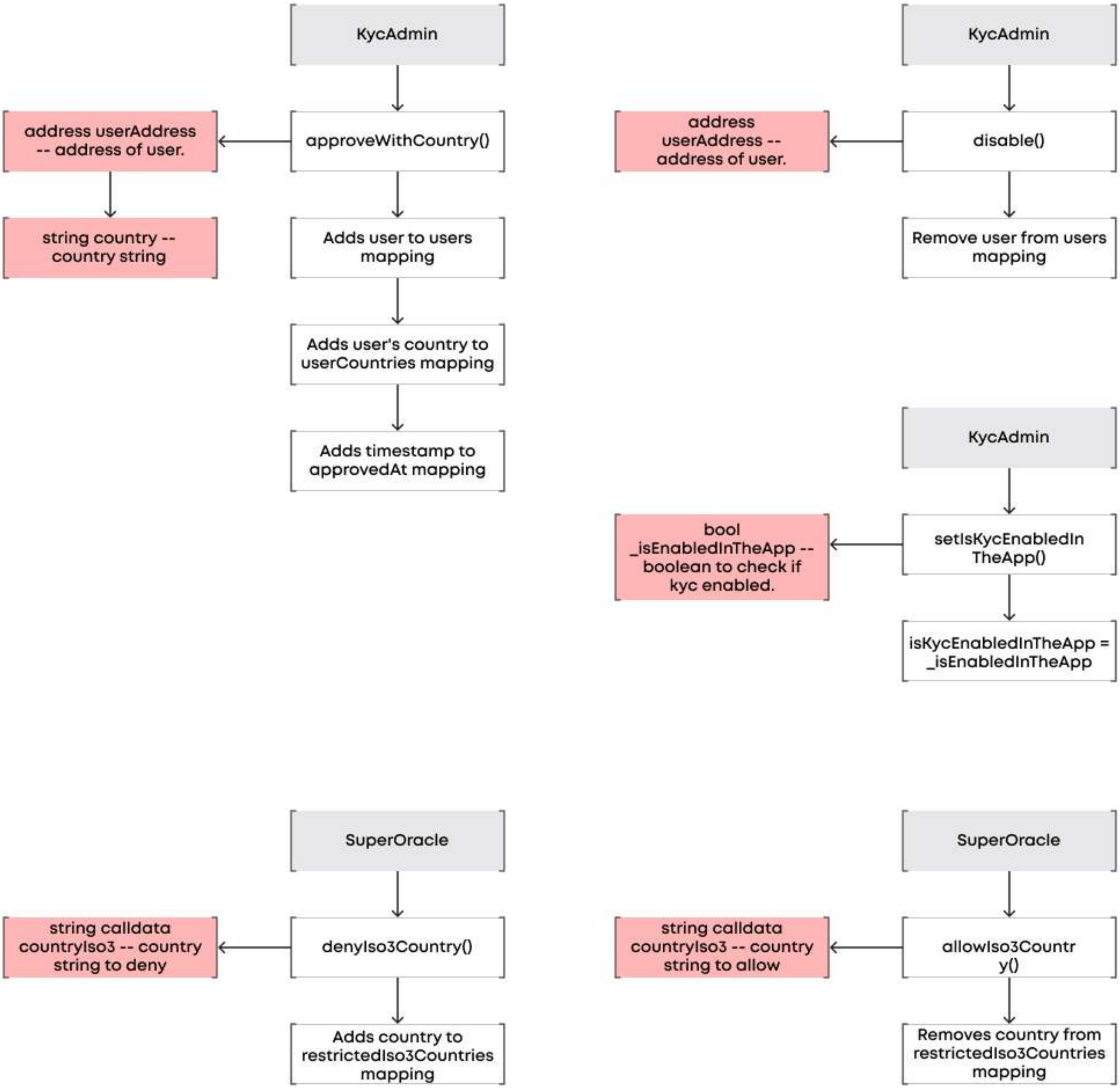

SuperOracle
setAssetPrice()
address asset -- address of asset to set price to.
uint256 price -- price for asset.
Checks that price > 0
Sets price and timestamp to assetPrice mapping


SuperOracle
addAsset()
address asset -- address of asset to set price to.
uint256 price -- price for asset.
Checks that asset is not added already
Sets price and timestamp to assetPrice mapping


SuperOracle
removeAsset()
address asset -- address of asset to remove.
Checks that asset is in assetPrice mapping
Deletes asset from assetPrice mapping

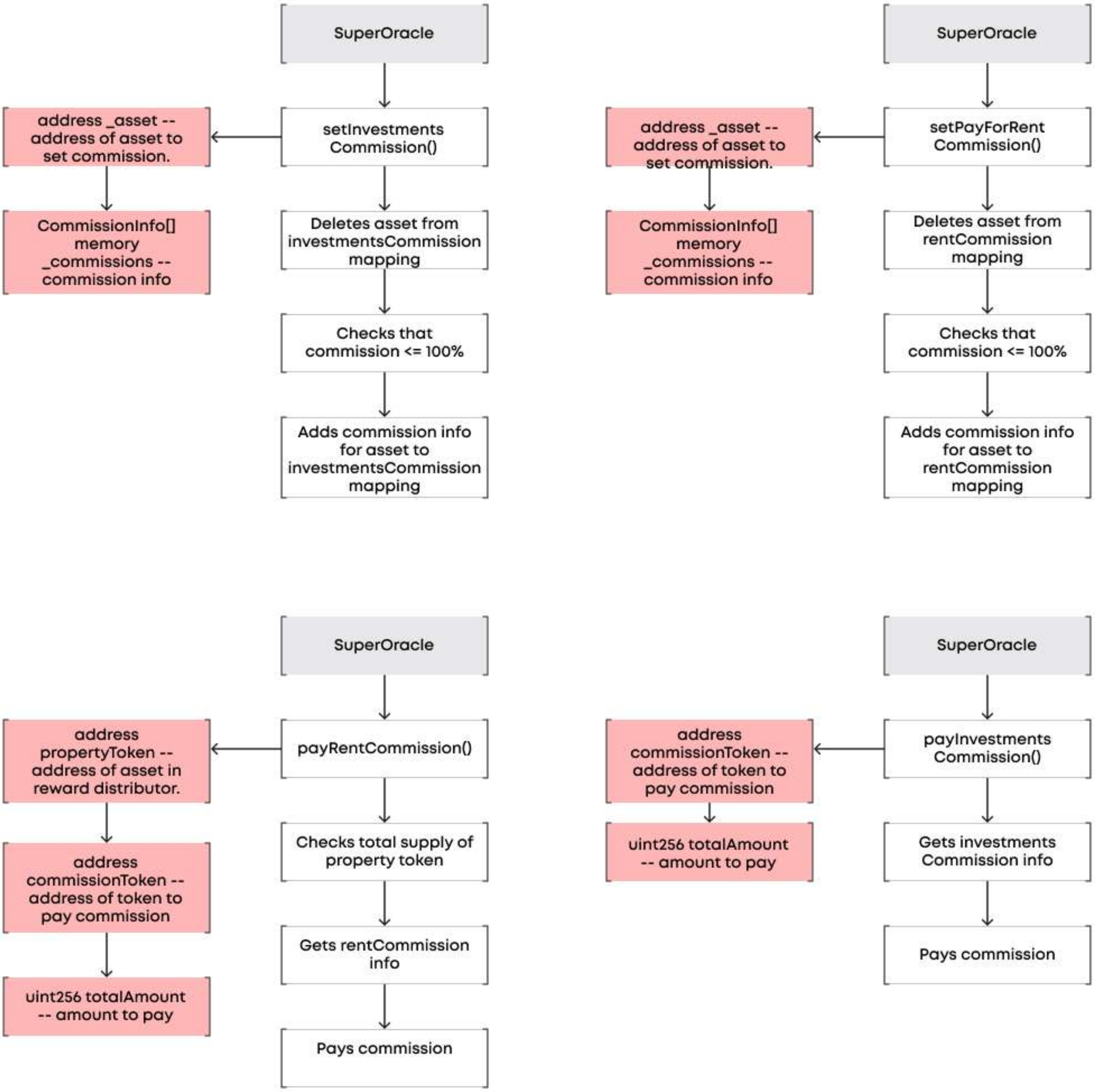
KycStore.sol

The KycStore contract is responsible for storing user data that has passed KYC and has been approved by an administrator. This data is required for a user to invest in an Asset.


KycAdmin
approveWithCountry()
address userAddress -- address of user.
string country -- country string
Adds user to users mapping
Adds user's country to userCountries mapping
Adds timestamp to approvedAt mapping
KycAdmin
disable()
address userAddress -- address of user.
Remove user from users mapping
KycAdmin
setIsKycEnabledInTheApp()
bool _isEnabledInTheApp -- boolean to check if kyc enabled.
isKycEnabledInTheApp = _isEnabledInTheApp
SuperOracle
denyIso3Country()
string calldata countryIso3 -- country string to deny
Adds country to restrictedIso3Countries mapping
SuperOracle
allowIso3Country()
string calldata countryIso3 -- country string to allow
Removes country from restrictedIso3Countries mapping

CommissionsDistributor.sol

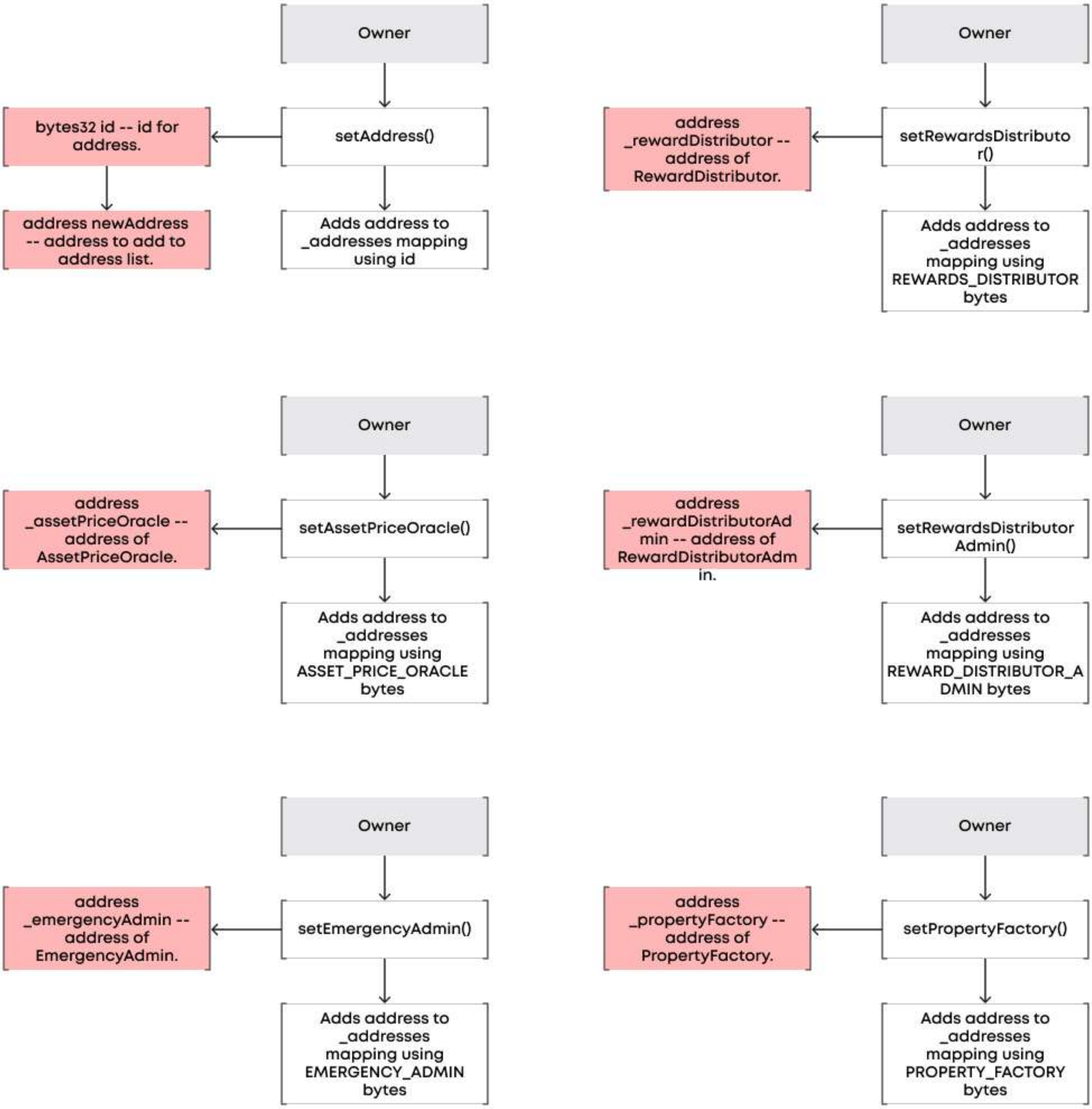
The CommissionsDistributor contract is responsible for managing and distributing commissions for investment rewards and fees within the protocol. It provides a centralized interface for administrators to track and distribute commissions, enabling efficient and transparent management of the protocol's financial operations.


SuperOracle
setInvestments Commission()
address _asset -- address of asset to set commission.
CommissionInfo[] memory _commissions -- commission info
Deletes asset from investmentsCommission mapping
Checks that commission <= 100%
Adds commission info for asset to investmentsCommission mapping
SuperOracle
setPayForRent Commission()
address _asset -- address of asset to set commission.
CommissionInfo[] memory _commissions -- commission info
Deletes asset from rentCommission mapping
Checks that commission <= 100%
Adds commission info for asset to rentCommission mapping
SuperOracle
payRentCommission()
address propertyToken -- address of asset in reward distributor.
address commissionToken -- address of token to pay commission
uint256 totalAmount -- amount to pay
Checks total supply of property token
Gets rentCommission info
Pays commission
SuperOracle
payInvestments Commission()
address commissionToken -- address of token to pay commission
uint256 totalAmount -- amount to pay
Gets investments Commission info
Pays commission

BINARYX

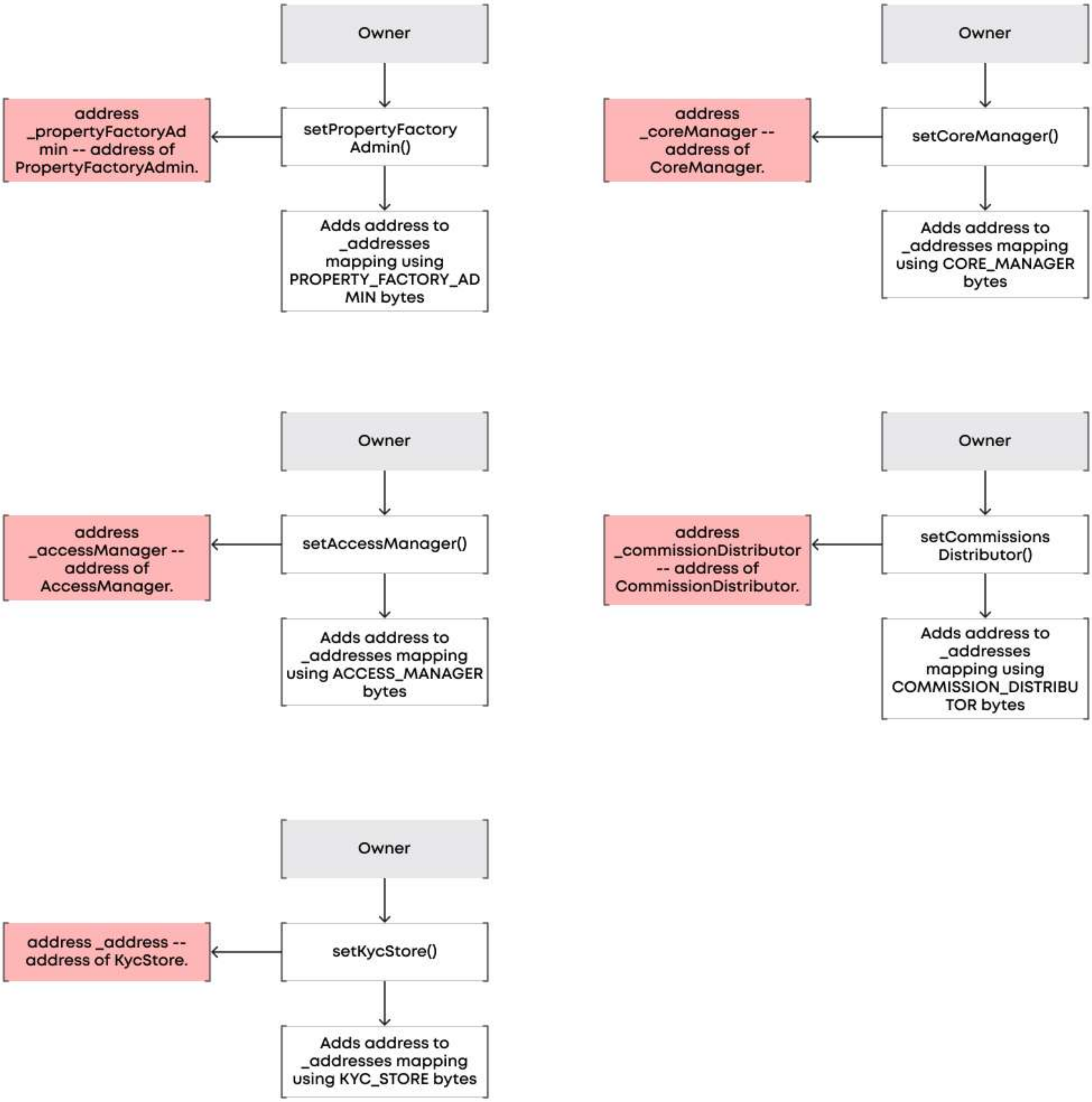
AddressesProvider.sol

The AddressesProvider contract serves as a central registry for all of the protocol's smart contracts. It provides a standardized interface for accessing the addresses of each contract, ensuring that contracts can interact with one another seamlessly.


Owner
setAddress()
bytes32 id -- id for address.
address newAddress -- address to add to address list.
Adds address to _addresses mapping using id
Owner
setRewardsDistributor()
address _rewardDistributor -- address of RewardDistributor.
Adds address to _addresses mapping using REWARDS_DISTRIBUTOR bytes
Owner
setAssetPriceOracle()
address _assetPriceOracle -- address of AssetPriceOracle.
Adds address to _addresses mapping using ASSET_PRICE_ORACLE bytes
Owner
setRewardsDistributorAdmin()
address _rewardDistributorAdmin -- address of RewardDistributorAdmin.
Adds address to _addresses mapping using REWARD_DISTRIBUTOR_ADMIN bytes
Owner
setEmergencyAdmin()
address _emergencyAdmin -- address of EmergencyAdmin.
Adds address to _addresses mapping using EMERGENCY_ADMIN bytes
Owner
setPropertyFactory()
address _propertyFactory -- address of PropertyFactory.
Adds address to _addresses mapping using PROPERTY_FACTORY bytes

AddressesProvider.sol

The AddressesProvider contract serves as a central registry for all of the protocol's smart contracts. It provides a standardized interface for accessing the addresses of each contract, ensuring that contracts can interact with one another seamlessly.


Owner
address _propertyFactoryAdmin -- address of PropertyFactoryAdmin.
setPropertyFactoryAdmin()
Adds address to _addresses mapping using PROPERTY_FACTORY_ADMIN bytes
Owner
address _coreManager -- address of CoreManager.
setCoreManager()
Adds address to _addresses mapping using CORE_MANAGER bytes
Owner
address _accessManager -- address of AccessManager.
setAccessManager()
Adds address to _addresses mapping using ACCESS_MANAGER bytes
Owner
address _commissionDistributor -- address of CommissionDistributor.
setCommissionsDistributor()
Adds address to _addresses mapping using COMMISSION_DISTRIBUTOR bytes
Owner
address _address -- address of KycStore.
setKycStore()
Adds address to _addresses mapping using KYC_STORE bytes

COMPLETE ANALYSIS

HIGH-1 ✓ **Resolved**

Users might receive less than they invested in case selling is rejected.

Asset.sol: rejectSelling(), line 141.
When a user invests in Asset the amount of buyToken, he might spend, depends on the price received from the Oracle. During the rejecting of selling, users might receive more or less of buyToken, depending on the price, at which they have invested.
For example:

- User1 wants to buy 5 Asset tokens. The price of buyToken is currently 2$. User1 spends 5 * 2 = 10 buyTokens.
- User2 wants to buy 10 Asset tokens. The price of Asset has changed and is currently 3$. User2 spends 10 * 3 = 30 buyTokens.
- Later, rejectSelling() is called and the amount of Asset is calculated based on totalSupply and the amount of buyToken on Asset's balance. Current total supply is 15 Asset tokens. There are 40 buyTokens on the Asset's balance.
- User1 will receive 5 * 40 / 15 = 13.3 buyTokens.
- User2 will receive 10 * 40 / 15 = 26.6 buyTokens.

Thus, user2 will receive less than he has invested.

Recommendation:

Verify that such functionality is correct and users might receive less than they have invested OR refund the exact amount which users invested. Notify users that they might receive less than they invested in case implementation should stay as is.

HIGH-2	✓ Resolved

Super Oracle can change the status of Asset without restriction.

Asset.sol: setStatus().

Address with role Super Oracle is able to change the status of Asset. This can affect the total flow of the protocol.

For example, in case of investing has been successful and the status has been changed to SoldOut, confirmSelling() can be called, and all collected buyTokens will be transferred from Asset to Seller, and the status will be changed to Confirmed.

However, after that, the status can be changed to Active or NotSoldOut making a rejectSelling() function callable and allowing Super Oracle to call it and burn Asset tokens from the balances of buyers. Though only Super Oracle can change the status of asset, such functionality leaves a dangerous backdoor on the contract. Which is why it is recommended to restrict this functionality, for example, by restricting from which to which state can the status be changed.

Recommendation:

Consider adding a restriction on the way, Super Oracle can change the status of the Asset.

MEDIUM-1	✓ Resolved

Transfer is not validated.

Asset.sol: invest(), line 80; confirmSelling(), line 125; rejectSelling(), line 147;
RewardsDistributor.sol: safeRewardTokenTransfer(), line 236,238; payForRent(), line 249
Transferring is performed with a regular transfer() method from the OpenZeppeling IERC20 interface without validation if transfer is successful. In order to ensure the security of the transfer it is recommended to validate the success of the transfer call. Thus, it is recommended to use SafeERC20 library and replace transfer() with safeTransfer() function.

Recommendation:

Use SafeERC20 library.

LOW-1	✓ Resolved

Fractional amount of asset can't be bought.

Asset.sol: invest().
When a user wants to invest in Asset, he provides an amount without decimals (e.g 1, 10, 12, etc). Decimals are added to the provided amount later. Thus, a fractional amount of Asset can't be passed. In case an initial value of `leftToBuy` was set to a fractional amount in the constructor, a fraction couldn't be bought. Due to this, the status of Asset wouldn't be changed to SoldOut in invest(). However, the issue is marked as low, since Super Oracle will still be able to set SoldOut status with setStatus().

Recommendation:

Verify that `leftToBuy` won't contain fraction OR allow users to provide value with fraction in invest() function.

LOW-2	✓ Resolved

Same user might be pushed to the array of buyers.

Asset.sol: _afterTokenTransfer(), line 103.
When a user is pushed to an array of buyers, it is not validated if he is already in the array. Thus, an array can store the same users, who have invested or received Asset several times. Though the issue doesn't prevent contract from working correctly and rejectSelling() works fine even with repeatable elements, it is recommended to avoid adding the same users to array.

Recommendation:

Track if the user is already in the array. For example, with the usage of additional mapping, so that array buyers store only unique values.

LOW-3	✓ Resolved

Array of buyers might contain empty elements and is not always cleaned.

Asset.sol: rejectSelling().
When a refunded user is removed from an array of buyers, the value is removed with operator delete which simply sets the value of the element to zero address. Thus, the number of elements is not changed and the array might contain empty elements. In such cases, the most common approach is to swap the value of the element to remove the value of the last element in the array and pop the last element out of the array.
Also, in case the balance of buyer is 0, user is not removed from the array (lines 137-138).

Recommendation:

Remove empty elements from the array. Ensure that processed users are removed in all cases.

LOW-4	✓ Resolved

Change of balance is tracked for zero address.

Asset.sol: _afterTokenTransfer(), line 108.
When Asset tokens are minted or burnt, _afterTokenTransfer() is called and sender or recipient is passed as zero address. Thus, in RewardsDistributor.sol: onUserBalanceChanged(), info about zero address will be updated (in `userToRewards`and `userToRewardUniq`). Though the balance of zero address is equal to zero, updating information for it in RewardsDistributor.sol increases gas spendings for invest() and rejectSelling() functions.

Recommendation:

Do not call onUserBalanceChanged() for zero address.

LOW-5	✓ Resolved

Unlimited allowance.

Asset.sol: _payCommission(), line 157.
An unlimited allowance in buyToken is granted to commissionDistributor in _payCommission(). Though, CommissionsDistributor is a part of the protocol, this contract is upgradeable, so it is not recommended to grant an unlimited allowance.

Recommendation:

Approve only a specific amount of tokens, necessary to pay commission.

LOWEST-1	Unresolved

Custom errors should be used.

Starting from the 0.8.4 version of Solidity, it is recommended to use custom errors instead of storing error message strings in storage and use “require” statements. Using custom errors is more efficient in terms of gas spending and increases code readability.

Recommendation:

Use custom errors.

LOWEST-2	Unresolved

Stick to solidity style guide.

It is better to stick to the style guide, it helps to read smart contracts and makes it easier to search for and interact with a smart contract.

Recommendation:

Stick to solidity style guide.

LOWEST-3	✓ Resolved

Function can be marked as pure.

BNRXToken.sol: getV(), line 28 can be marked pure.

Recommendation:

Change the modifier to pure.

LOWEST-4	✓ Resolved

Token implementation should be clarified.

BNRXToken.sol:
VERIFIED
1) It should be noted that the token is upgradeable. Most of the centralized exchanges might not list token upgradeable functionality. The upgradeability of the token is not trustworthy for users as well.
RESOLVED
2) Token inherits ERC20BurnableUpgradeable, which allows anyone to burn their tokens. Though burnable functionality is not a security issue by itself, it might be used by malicious actors to affect the price of a token.
RESOLVED
3) Token inherits PausableUpgradeable, but none of the operations can be paused: transfer, burn, and mint can't be paused.

Recommendation:

Verify that the token should be upgradeable. Verify that token should be burnable and anyone should be able to burn their tokens OR consider restricting burn function, so that only specific addresses can burn. Remove PausableUpgradeable or add the ability to pause necessary operations.

LOWEST-5	✓ Resolved

Lack of events.

KycStore.sol: disable(), setIsKycEnabledInTheApp(), denyIso3Country(), allowIso3Country().
AccessManager.sol: addKycOracle(), removeKycOracle(), removeSuperOracle(), addSuperOracle().
AssetPriceOracle.sol: setAssetPrice(), addAsset(), removeAsset().
RewardsDistributor.sol: setClaimReceiver().
In order to keep track of historical changes in storage variables, it is recommended to emit events on every change in setters.

Recommendation:

Emit event in setters.

LOWEST-6	✓ Resolved

Unused mapping.

RewardsDistributor.sol: `claimReceiver`.
Mapping is defined, and there is a setter for it. However the values from mapping are not used in the code.

Recommendation:

Use mapping or verify that it is necessary for future updates.

LOWEST-7	✓ Resolved

Duplicated code.

Asset.sol: setStatus(), lines 114-115.
In the function given lines checks that msg.sender is SuperOracle, but in contrast there exists a modifier onlySuperOracle, that checks the same.

Recommendation:

Use a modifier instead.

LOWEST-8	✓ Resolved

Users cannot unstake their tokens manually.

Asset.sol: rejectSelling()
Only superOracle can unstake users' tokens. According to the info from the website, the users can unstake anytime. If the sale round for assets has an end time, it is better to keep user funds until the end of the round of sales. Thus, users will be able to see how much time their funds are blocked on your smart contract. At the moment, if the superOracle does not use the function, users will not be able to get their tokens back.

Recommendation:

Use the time of sale round in order to give users an understanding of how long their funds are blocked, and when they can withdraw their funds themselves.

		contracts\v3 **AccessManager.sol** **AddressesProvider.sol** **Asset.sol** **AssetPriceOracle.sol**
✓	Re-entrancy	Pass
✓	Access Management Hierarchy	Pass
✓	Arithmetic Over/Under Flows	Pass
✓	Delegatecall Unexpected Ether	Pass
✓	Default Public Visibility	Pass
✓	Hidden Malicious Code	Pass
✓	Entropy Illusion (Lack of Randomness)	Pass
✓	External Contract Referencing	Pass
✓	Short Address/Parameter Attack	Pass
✓	Unchecked CALL Return Values	Pass
✓	Race Conditions/Front Running	Pass
✓	General Denial Of Service (DOS)	Pass
✓	Uninitialized Storage Pointers	Pass
✓	Floating Points and Precision	Pass
✓	Tx.Origin Authentication	Pass
✓	Signatures Replay	Pass
✓	Pool Asset Security (backdoors in the underlying ERC-20)	Pass

		contracts\v3 **BNRXToken.sol** **CommissionsDistributor.sol** **CoreManager.sol** **KycStore.sol**
✓	Re-entrancy	Pass
✓	Access Management Hierarchy	Pass
✓	Arithmetic Over/Under Flows	Pass
✓	Delegatecall Unexpected Ether	Pass
✓	Default Public Visibility	Pass
✓	Hidden Malicious Code	Pass
✓	Entropy Illusion (Lack of Randomness)	Pass
✓	External Contract Referencing	Pass
✓	Short Address/Parameter Attack	Pass
✓	Unchecked CALL Return Values	Pass
✓	Race Conditions/Front Running	Pass
✓	General Denial Of Service (DOS)	Pass
✓	Uninitialized Storage Pointers	Pass
✓	Floating Points and Precision	Pass
✓	Tx.Origin Authentication	Pass
✓	Signatures Replay	Pass
✓	Pool Asset Security (backdoors in the underlying ERC-20)	Pass

contracts\v3
Oracle.sol
OracleFactory.sol
PropertyFactory.sol
RewardsDistributor.sol

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

		contracts\v3 UiProvider.sol UsdtfToken.sol
✓	Re-entrancy	Pass
✓	Access Management Hierarchy	Pass
✓	Arithmetic Over/Under Flows	Pass
✓	Delegatecall Unexpected Ether	Pass
✓	Default Public Visibility	Pass
✓	Hidden Malicious Code	Pass
✓	Entropy Illusion (Lack of Randomness)	Pass
✓	External Contract Referencing	Pass
✓	Short Address/Parameter Attack	Pass
✓	Unchecked CALL Return Values	Pass
✓	Race Conditions/Front Running	Pass
✓	General Denial Of Service (DOS)	Pass
✓	Uninitialized Storage Pointers	Pass
✓	Floating Points and Precision	Pass
✓	Tx.Origin Authentication	Pass
✓	Signatures Replay	Pass
✓	Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BINARYX TEAM

AssetPriceOracle

deploys

✓ with valid params (8750ms)

addAsset

✓ is denied for it's owner (158ms)

✓ is allowed by super oracle (57ms)

KycStore

✓ deploys

✓ returns non existing (40ms)

✓ approves and return existing (104ms)

✓ disable and return as disabled (65ms)

✓ empty country should be allowed (62ms)

✓ Rest countries country should be allowed (41ms)

✓ denies and allows country (123ms)

RewardDistributor

✓ Deploy RewardDistributor (138ms)

✓ Add pool (263ms)

✓ Pay for rent (1049ms)

✓ Claim reward flow (3 users) with more then 1 emission point (1308ms)

✓ Claim reward flow (3 users) in one emission point (1276ms)

✓ Claimable reward (3 users) with more then 1 emission point (1269ms)

✓ Claimable reward (2 users) in one emission point (675ms)

✓ User should get their reward even if they claimed already (942ms)

AccessManager

deploying

✓ with default values (12739ms)

super oracles

✓ can be added and removed (89ms)

kyc oracles

✓ can be added and removed (113ms)

```
    Asset
  ✓ Deploy Asset (503ms)
  ✓ Invest in Asset without KYC should throw (279ms)
  ✓ Invest in Asset with KYC should pass (383ms)
  ✓ Change status (237ms)
  ✓ Confirm Selling with commissions (631ms)
  ✓ Reject Selling with distribution invest USDT (737ms)
  ✓ should throw transfer without KYC (373ms)
  ✓ should transfer with KYC (409ms)
    UiProvider
    list (public)
  ✓ returns empty results (42ms)
  ✓ returns all results (2517ms)
  ✓ return needed fields (485ms)
    getMyRewards
  ✓ return needed fields + rewards (1705ms)
  ✓ do not stack records (740ms)

    34 passing (38s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
AccessManager.sol	75	50	87.5
AddressesProvider.sol	83.33	45.83	83.33
Asset.sol	95.89	48.33	90.48
AssetPriceOracle.sol	35.71	33.33	50
BNRXToken.sol	0	0	0
CommissionsDistributor.sol	83.33	45.45	72.73
CoreManager.sol	55.81	28.13	46.67
KycStore.sol	77.78	44.44	80
Oracle.sol	0	0	0
OracleFactory.sol	0	12.5	20

FILE	% STMTS	% BRANCH	% FUNCS
PropertyFactory.sol	87.5	50	83.33
RewardsDistributor.sol	91.59	63.75	75
UiProvider.sol	69.23	100	83.33
UsdtfToken.sol	66.67	100	66.67
All files	58.7	44.41	59.93

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

AccessManager:

- ✓ Should not initialize twice
- ✓ Should get version
- ✓ User should not be KycOracle by default
- ✓ User should not be SuperOracle by default
- ✓ Admin should add and remove SuperOracle
- ✓ Not admin should not add SuperOracle
- ✓ Not admin should not remove SuperOracle
- ✓ Admin should add and remove KycOracle
- ✓ Not admin should not add KycOracle
- ✓ Not admin should not remove KycOracle

AddressesProvider:

- ✓ Should not initialize twice
- ✓ Should get version

Owner should set parameters

- ✓ Should set new address by ID
- ✓ Should set REWARDS_DISTRIBUTOR
- ✓ Should set ASSET_PRICE_ORACLE
- ✓ Should set REWARD_DISTRIBUTOR_ADMIN
- ✓ Should set EMERGENCY_ADMIN
- ✓ Should set PROPERTY_FACTORY
- ✓ Should set PROPERTY_FACTORY_ADMIN
- ✓ Should set CORE_MANAGER
- ✓ Should set ACCESS_MANAGER
- ✓ Should set COMMISSION_DISTRIBUTOR
- ✓ Should set KYC_STORE

User (not owner) should not set parameters

- ✓ Should not set new address by ID
- ✓ Should not set REWARDS_DISTRIBUTOR
- ✓ Should not set ASSET_PRICE_ORACLE
- ✓ Should not set REWARD_DISTRIBUTOR_ADMIN
- ✓ Should not set EMERGENCY_ADMIN
- ✓ Should not set PROPERTY_FACTORY

- ✓ Should not set PROPERTY_FACTORY_ADMIN
- ✓ Should not set CORE_MANAGER
- ✓ Should not set ACCESS_MANAGER
- ✓ Should not set COMMISSION_DISTRIBUTOR
- ✓ Should not set KYC_STORE

Asset

Main

- ✓ Get version
- ✓ Set payment token range
- ✓ Set ChainLink price oracle
- ✓ Set sale end date
- ✓ Invest (73ms)
- ✓ Tokens sold out (64ms)
- ✓ Confirm selling (83ms)
- ✓ Reject selling (164ms)
- ✓ Reject selling as user (141ms)

Updating

- ✓ Status (186ms)
- ✓ Asset info
- ✓ Asset location data
- ✓ Documents

Revert

- ✓ When try to initialize contract again
- ✓ When not admin try to set token range
- ✓ When not admin try to set ChainLink oracle
- ✓ When not admin try to set sale end
- ✓ When try to set sale end if time is less that current time
- ✓ When status is not Active
- ✓ When amount is 0
- ✓ When amount is more that left to buy
- ✓ When rewardsDistributor is not set
- ✓ When oracle is not set
- ✓ When price is not set (41ms)
- ✓ When user does not pass kyc
- ✓ When recipient does not pass kyc (69ms)
- ✓ When not Super oracle try to change status
- ✓ When try to confirm sale if sender is not super oracle

- ✓ When try to confirm sale if status is not SoldOut
- ✓ When try to confirm sale if seller is zero address (67ms)
- ✓ When commissionDistributor is not set (76ms)
- ✓ When try to reject sale if sender is not super oracle
- ✓ When try to reject sale if seller is status is not Active or NotSoldOut
- ✓ When try to reject sale if start index > end index
- ✓ When try to reject sale if end index > buyers
- ✓ When try to reject sale as user if status is not Active
- ✓ When try to reject sale as user if sale has not ended
- ✓ When try to reject sale as user if user is not buyer
- ✓ When not super oracle try to update asset info
- ✓ When not super oracle try to update asset location
- ✓ When not super oracle try to add documents
- ✓ When not super oracle try to update documents
- ✓ When documentIndex > documents length

Audit check High-1 (fixed)

User gets less tokens when rejected if price change

- ✓ Asset price is 2$
- ✓ Invest as first user for 5 asset tokens (10 buyTokens) (38ms)
- ✓ Asset price changed to 3$
- ✓ Invest as second user for 10 asset tokens (30 buyTokens)
- ✓ Reject selling for users
- ✓ Check, that initial buyToken user balances == user balances now

AssetPriceOracle

Main

- ✓ Get version
- ✓ Set asset price
- ✓ Set multiple assets price
- ✓ Add asset
- ✓ Add multiple assets
- ✓ Remove asset

Revert

- ✓ When try to initialize contract again
- ✓ When not super oracle try to set asset price
- ✓ When not super oracle try to add asset
- ✓ When not super oracle try to remove asset
- ✓ When asset price is 0
- ✓ When assets and price should be same length

- ✓ When asset already exist
- ✓ When add assets and price should be same length
- ✓ When asset does not exist

BNRXToken

Mai

- ✓ Get version
- ✓ Approve
- ✓ Increase/Decrease allowance
- ✓ Transfer
- ✓ TransferFrom
- ✓ Pause/unpaus

Revert

- ✓ When try to initialize contract again
- ✓ When not owner try to pause/unpause
- ✓ When try to transfer when contract is paused
- ✓ When try to transferFrom when contract is paused
- ✓ When try to increase/decrease allowance when contract is paused
- ✓ When try to approve tokens when contract is paused

CommissionsDistributor

Getters

- ✓ Should get version
- ✓ Should get rent commission

Setters

- ✓ Should set investments commission
- ✓ Shouldn't set investments commission by everyone but admin
- ✓ Shouldn't set investments commission with invalid percent
- ✓ Shouldn't set rent commission with invalid percent
- ✓ Shouldn't set rent commission by everyone but admin

Pay commission operations

- ✓ Should pay investments commission (41ms)
- ✓ Should pay commission with fixed amount (39ms)
- ✓ Shouldn't pay rent commission by everyone but rewardDistributor
- ✓ Shouldn't pay commission when amount less than commission
- ✓ Should revert when commission is more than amount

CoreManager

Getters

- ✓ Should get version

Emergency operations

- ✓ Should revert emergency call without reason string
- ✓ Should emergency call
- ✓ Should revert with error message
- ✓ Shouldn't make emergency call by everyone but emergency admin

Pay rent

- ✓ Should pay rent (339ms)
- ✓ Shouldn't pay rent without needed allowance (230ms)
- ✓ Should pay rent if pool already initialized (269ms)
- ✓ Shouldn't pay rent by everyone but coreManagerAdmin (223ms)

Listing asset with post updates

- ✓ Should list asset (164ms)
- ✓ Shouldn't listAsset by everyone but coreManagerAdmin (84ms)
- ✓ Should update docs (174ms)
- ✓ Shouldn't update docs by everyone but coreManagerAdmin (158ms)
- ✓ Should add docs (177ms)
- ✓ Shouldn't add docs by everyone but coreManagerAdmin (163ms)
- ✓ Should update asset info (308ms)
- ✓ Shouldn't update asset info by everyone but coreManagerAdmin (157ms)
- ✓ Should update asset location data (167ms)
- ✓ Shouldn't update asset location data by everyone but coreManagerAdmin (165ms)
- ✓ Should update all asset data (234ms)
- ✓ Shouldn't update all asset data (151ms)

KycStore:

- ✓ Should not initialize twice
- ✓ Should get version
- ✓ KycAdmin should approve with country
- ✓ Not KycAdmin should not approve with country
- ✓ KycAdmin should disable user

- ✓ Not KycAdmin should not disable user
- ✓ KycAdmin should set is kycEnabled
- ✓ Not KycAdmin should not set is kycEnabled
- ✓ SuperOracle should deny and allow Iso3Country
- ✓ Not SuperOracle should not deny Iso3Country
- ✓ Not SuperOracle should not allow Iso3Country
- ✓ User should get isOperable

Oracle

Main

- ✓ Set status
- ✓ Set oracle type
- ✓ Set owner
- ✓ Add member
- ✓ Update document

Revert

- ✓ When owner is 0 address
- ✓ When documentIndex > documents length
- ✓ When not owner try to set status
- ✓ When not owner try to set oracle type
- ✓ When not owner try to set owner
- ✓ When not owner try to add member
- ✓ When not owner try to add document
- ✓ When not owner try to update document

OracleFactory

Main

- ✓ Deploy oracle

Revert

- ✓ When try to initialize contract again
- ✓ When not super oracle try to deploy oracle
- ✓ When owner address is 0

PropertyFactory

Main

- ✓ Get version
- ✓ Get beacon
- ✓ Get implementation
- ✓ Deploy oracle (43ms)

Revert

- ✓ When try to initialize contract again
- ✓ When not super oracle try to deploy oracle

RewardsDistributor

Initialization

- ✓ Should initialize correctl
- ✓ Shouldn't initialize when addressesProvider and rewardToken are zero addresses

Getters

- ✓ Should get version
- ✓ Should get reward token
- ✓ Shouldn't check onUserBalanceChanged with non-existent pool

Pool operations

- ✓ Should add pool
- ✓ Shouldn't add pool twice
- ✓ Should add pool only by owner
- ✓ Should initialize pool
- ✓ Shouldn't initialize pool by everyone but owner
- ✓ Shouldn't initialize already initialized pool
- ✓ Should return 0 when call calculateActualEmissionPointPerPool if schedule has zero length

Rewards operations

- ✓ Should pay rent (108ms)
- ✓ Shouldn't pay rent by everyone but owner (75ms)
- ✓ Shouldn't pay rent with non-initialized pool (69ms)
- ✓ Shouldn't pay rent with 0 amount (72ms)
- ✓ Shouldn't pay rent with endTime less than startTime (72ms)
- ✓ Should claim rewards (2 user, 1 emission point) (203ms)
- ✓ Claimable rewards should be [0] (96ms)
- ✓ Should add base claimable when transferring tokens (158ms)
- ✓ Should claim rewards (1 user, 1 emission point) (170ms)
- ✓ Should claim rewards (2 user, 2 emission point) (250ms)
- ✓ Should return 0 claimable rewards if 0 emission points (111ms)
- ✓ Shouldn't pay rent if emissions invalid setted (147ms)
- ✓ Shouldn't claim rewards if there is no added pools (41ms)

```
    ✓ Shouldn't pay rent commission if there is no added pools (71ms)
  UiProvider
      Asset operations
    ✓ Should get asset info and asset sell progress (294ms)
      Rewards operations
    ✓ Should get rewards and user total rewards (418ms)
  UsdtfToken
      Main
    ✓ Decimals
    ✓ Get limit amount
    ✓ Get limit time
      Mint
    ✓ User mint
    ✓ Demo mint
      Admin functions
    ✓ Change amount limit
    ✓ Change time limit
      Revert
    ✓ When user has already minted
    ✓ When not admin try to demo mint
    ✓ When not admin try to change amount limit
    ✓ When not admin try to change time limit

      217 passing (12s)
```

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
AccessManager.sol	100	100	100
AddressesProvider.sol	100	100	100
Asset.sol	97.6	89.68	96.55
AssetPriceOracle.sol	100	100	100
BNRXToken.sol	100	100	100
CommissionsDistributor.sol	100	100	100
CoreManager.sol	100	100	100
KycStore.sol	100	100	100
Oracle.sol	100	100	100
OracleFactory.sol	100	100	100

FILE	% STMTS	% BRANCH	% FUNCS
PropertyFactory.sol	100	100	100
RewardsDistributor.sol	99.06	92.86	100
UiProvider.sol	100	100	100
UsdtfToken.sol	100	100	100
All files	**99.8**	**98.8**	**99.8**

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.