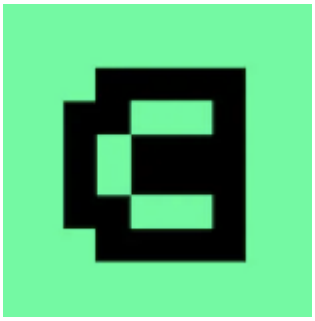
Canto Application Specific Dollars and Bonding Curves for 1155s



Scope

The code under review can be found within the C4 Canto Application Specific Dollars and Bonding Curves for 1155s repository.

Summary

Findings

QA issues	Issues	Severity
L-01	Mapping shareBondingCurves is not updated when creating new share	Low
L-02	Missing `shareData[_id].creator != msg.sender` check in sell() function	Low
L-03	Revert with an error message instead of letting the underflow revert in sell() function	Low
L-04	Share holder can spam off-chain tracking with 0 amount event emission in fuction claimHolderFee()	Low
L-05	Incorrect fee is returned by getNFTMintingPrice()	Low
N-01	Missing natspec dev comment to approve underlying tokens to Market.sol for transfer	Non-Critical
N-02	Rename LinearBondingCurve.sol contract to InverseLogarithmicCurve.sol contract	Non-Critical

Gas Optimizations

Gas Optimizations	Issues	Instances
G-01	Remove else block to save gas	1
G-02	Remove redundant mapping shareBondingCurves to save gas	1

Gas Optimizations	Issues	Instances
G-03	Parameter can be made calldata to save gas	1
G-04	Remove redundant check `shareData[_id].creator != msg.sender` to save gas	1

Analysis Report

Findings

[L-01] Mapping shareBondingCurves is not updated when creating new share

Link to instance

The mapping shareBondingCurves is supposed to store the bonding curve being used for a shareID. But when createNewShare() is called, the function does not update the mapping for the shareID created. Although the mapping is not used anywhere in the current contract, if any external protocols or contracts use the value from that mapping, it will return an incorrect value i.e. address(0)

We can observe from Line 125-128 no update is made to the mapping.

```
File: Market.sol
117:     function createNewShare(
118:         string memory _shareName,
119:         address _bondingCurve,
120:         string memory _metadataURI
121:     ) external onlyShareCreator returns (uint256 id) {
122:         require(whitelistedBondingCurves[_bondingCurve], "Bonding
curve not whitelisted");
123:         require(shareIDs[_shareName] == 0, "Share already exists");
124:         id = ++shareCount;
125:         shareIDs[_shareName] = id;
126:         shareData[id].bondingCurve = _bondingCurve; //@audit
shareBondingCurves mapping is not updated here
127:         shareData[id].creator = msg.sender;
128:         shareData[id].metadataURI = _metadataURI;
129:         emit ShareCreated(id, _shareName, _bondingCurve, msg.sender);
130:     }
```

Solution: Update the mapping for that shareID to store the address of the bonding curve

[L-02] Missing `shareData[_id].creator != msg.sender` check in sell() function

Link to instance

Unlike the buy() function, the sell function() does not include a check to see if the msg.sender is not the creator of the shareID for which tokens are being sold.

As mentioned in the publicly known issues section in the README, **"This check can be circumvented easily (buying from a different address or buying an ERC1155 token on the secondary market and unwrapping it). The main motivation behind this check is to make clear that the intention of the system is not that creators buy a lot of their own tokens (which is the case for other bonding curve protocols), it is not a strict security check. "**

In our case, it would be to prevent the creator from mass selling tokens from his/her original address. Though market-making cannot be prevented (such as creator mass selling tokens to short on an exchange), the check should be implemented as done in the buy() function.

Solution:

```
File: Market.sol
178:     function sell(uint256 _id, uint256 _amount) external {
179:         require(shareData[_id].creator != msg.sender, "Creator cannot
sell");
180:         (uint256 price, uint256 fee) = getSellPrice(_id, _amount);
```

[L-03] Revert with an error message instead of letting the underflow revert in sell() function

Link to instance

In the line below from the sell() function we can observe that if the user does not have enough tokens to sell, the subtraction would underflow. This is not the best practice and is not recommended since it leaves the user without a reason while giving an underflow error.

```
File: Market.sol
188:         tokensByAddress[_id][msg.sender] -= _amount; // Would
underflow if user did not have enough tokens
```

Solution: Add a check to see if `amount >= tokensByAddress[_id][msg.sender]`. If not then revert with an insufficient balance error

[L-04] Share holder can spam off-chain tracking with 0 amount event emission in fuction claimHolderFee()

Link to instance

A holder with 0 holder fees can claim 0 amount and spam event emissions. This will spam the off-chain monitoring system with 0 amount claim entries. Though no risk, the call should be reverted in case `amount == 0` to ensure proper error handling and preventing event emission spamming.

```
File: Market.sol
271:     function claimHolderFee(uint256 _id) external {
272:         uint256 amount = _getRewardsSinceLastClaim(_id);
```

```
273:         rewardsLastClaimedValue[_id][msg.sender] =
shareData[_id].shareHolderRewardsPerTokenScaled;
274:         if (amount > 0) {
275:             SafeERC20.safeTransfer(token, msg.sender, amount);
276:         }
277:         emit HolderFeeClaimed(msg.sender, _id, amount);
278:     }
```

[L-05] Incorrect fee is returned by getNFTMintingPrice()

Impact

When minting or burning through mintNFT() and burnNFT(), the value for the fee to be charged is fetched from the getNFTMintingPrice() function. But the price fetched is incorrect since when getPriceAndFee() function is called on the LinearBondingCurve.sol contract, the parameter `shareCount` is passed as `shareData[_id].tokenCount` instead of `shareData[_id].tokenCount + 1`. Due to this incorrect price is returned leading to incorrect fees being determined. This allows users to mint/burn for a lesser fee than the expected, which will be a loss of fee rewards for the creator and platform.

Proof of Concept

Coded POC

Here are few points to know about the POC:

- The POC is a modification of an existing test function testGetBuyPrice()
- The POC shows that since the `shareData[_id].tokenCount` does not add 1 in the parameter, it leads to an incorrect price being retrieved and thus incorrect fee being determined from this price.
- This POC requires one small modification in the Market.sol contract i.e. to also return the price variable `priceForOne` when getNFTMintingPrice() is called. This will help us prove that the price returned is incorrect and thus the fee determined.
- Run the test using `forge test --match-test testIncorrectPrice -vvvvv`
- The test will fail (which is expected since incorrect prices are returned). In the stack traces we can observe the assert values that do not equate to the expected value.

```
File: Market.t.sol
105:     function testIncorrectPrice() public {
106:         testCreateNewShare();
107:         (uint256 priceOne, uint256 feeOne) =
market.getNFTMintingPrice(1, 1);
108:         (uint256 priceTwo, uint256 feeTwo) =
market.getNFTMintingPrice(1, 2);
109:         assertEq(priceOne, LINEAR_INCREASE);
110:         assertEq(priceTwo, priceOne + LINEAR_INCREASE * 2);
111:         assertEq(feeOne, priceOne / 10);
112:         assertEq(feeTwo, priceTwo / 10); // log2(2) = 1
113:     }
```

Tools Used

Manual Review

Recommended Mitigation Steps

Use `shareData[_id].tokenCount + 1` instead of `shareData[_id].tokenCount` as value in the parameter when making a call to getPriceAndFee() function

[N-01] Missing natspec dev comment to approve underlying tokens to Market.sol for transfer

Link to instance

The buy() function transfers the underlying tokens from the msg.sender to the Market.sol contract when purchasing tokens of a shareID. But for this the users need to approve the Market.sol with the underlying amount of tokens.

```
File: Market.sol
150:     /// @notice Buy amount of tokens for a given share ID
151:     // @dev User needs to approve the contract with price + fee or
more amount of tokens
152:     /// @param _id ID of the share
153:     /// @param _amount Amount of shares to buy
```

[N-02] Rename LinearBondingCurve.sol contract to InverseLogarithmicCurve.sol contract

Link to instance

The LinearBondingCurve.sol contract does not act as a linear bonding curve but instead an inverse logarithmic curve due to the formula `0.1 / log2(shareCount)` being used. This contract can be confusing to understand initially since the contract name is differing from the implementation.

Gas Optimizations

[G-01] Remove else block to save gas

Link to instance

Function execution cost: 860 - 848 = 12 gas saved (per call)

Instead of this:

```
File: LinearBondingCurve.sol
27:
28:     function getFee(uint256 shareCount) public pure override returns
(uint256) {
29:         uint256 divisor;
```

```
30:         if (shareCount > 1) {
31:             divisor = log2(shareCount);
32:         } else {
33:             divisor = 1;
34:         }
35:         // 0.1 / log2(shareCount)
36:         return 1e17 / divisor;
37:     }
```

Use this:

```
File: LinearBondingCurve.sol
27:
28:     function getFee(uint256 shareCount) public pure override returns
(uint256) {
29:         uint256 divisor = 1;
30:         if (shareCount > 1) {
31:             divisor = log2(shareCount);
32:         }
33:         // 0.1 / log2(shareCount)
34:         return 1e17 / divisor;
35:     }
```

[G-02] Remove redundant mapping shareBondingCurves to save gas

Link to instance

The mapping shareBondingCurves is not used anywhere in the contract and can thus be removed.

Deployment cost: 2499354 - 2487364 = 11990 gas saved

Remove the mapping shareBondingCurves below:

```
File: Market.sol
45:     /// @notice Stores the bonding curve per share
46:     mapping(uint256 => address) public shareBondingCurves;
```

[G-03] Parameter can be made calldata to save gas

Link to instance

Parameters that are only read can be marked calldata.

Function execution cost: 10457 - 10068 = 389 gas saved (per call)

Instead of this:

```
File: Market.sol
115:     function createNewShare(
116:         string memory _shareName,
117:         address _bondingCurve,
118:         string memory _metadataURI
```

Use this:

```
File: Market.sol
115:     function createNewShare(
116:         string calldata _shareName,
117:         address _bondingCurve,
118:         string calldata _metadataURI
```

[G-04] Remove redundant check `shareData[_id].creator != msg.sender` to save gas

Link to instance

The check `require(shareData[_id].creator != msg.sender, "Creator cannot buy");` on Line 151 can be circumvented by the creator by using another address as mentioned in the README's publicly known issues section. This check has no use if it can be circumvented and should be removed.

Deployment cost: 2499354 - 2479531 = 19823 gas saved

Function execution cost: 157773 - 157550 = 223 gas saved (per call)

Remove the check shown below:

```
File: Market.sol
151:  require(shareData[_id].creator != msg.sender, "Creator cannot buy");
```

Analysis Report

Approach taken in evaluating the codebase

Time spent on this audit: 2 days

Day 1

- Reviewed all four contracts in scope
- Added inline bookmarks
- Diagramming features and mental models for mechanisms

Day 2

- Asking sponsor questions to strengthen understanding of the codebase and certain mechanisms

- Writing reports for all issues
- Spending more time reviewing Market.sol and LinearBondingCurve.sol contracts

Architecture recommendations

Protocol Structure

The codebase can be divided into two protocols, namely, the asD protocol and the 1155Tech protocol. The asD protocol serves the 1155Tech protocol by providing the application specific dollars (asDs) created as the underlying currency for buying/selling shares and minting/burning ERC1155 tokens.

What's unique?

- Using ERC1155 standard for art - The ERC1155 standard is not as widely used as ERC721. By using 1155, the team is providing artists with new opportunities to showcase their art and become a creator of a share. This further incentivises the artwork only since the creator earns fees and not the original amount of asD paid for the art.
- Using asDs - Enabling the creation of stablecoins permissionlessly has allowed any application to create stablecoins for their use case and building a market on top of it by using Market.sol or their specific contracts.
- Incentivizing users - Retaining users is the most important aspect when building a art protocol since the users need a reason to stay and hold their ERC1155 tokens. In the 1155Tech protocol, this is done by providing the users with accrued fees, which scale as more users interact with the protocol.
- Restricting and whitelisting creators - Being permisionless is great but not at the cost of unnecessary spam creations of new shareIDs. Adding a whitelist to restrict certain creators brings the protocol into a stable state that allows the 1155Tech protocol to onboard users.
- Allowing share creators to have a different metadata uri than the original base uri allows the creators to showcase their own artwork using their own IPFS bucket instead of going through the admin of the contract to store the media in their IPFS solution.

What ideas can be incorporated?

- Providing rewards for holding NFTs of specific shareIds
- Having X amount of shares across X shareIds qualifies for fee rewards
- Allowing platform owner, creator of share and holder of share to withdraw only certain amount of fees instead of claiming and sending them the whole balance
- Allowlist users as well for allowlist and public phase minting
- Supporting more bonding curves

Codebase quality analysis

The codebase is quite straightforward to understand after reading the README.

Here are certain aspects which can be improved:

1. Adding more tests to test different execution path combinations such as user burns an NFT and tries to sell or mints an NFT and tries to buy more. This will ensure no edge cases or accounting is left out.
2. Adding more documentation for devs/security researcher to understand functionality of mint() and burn() functions.

Other than this, the most important aspects of the codebase have been implemented correctly such as ensuring correct fee accounting while buying, selling, minting, burning and updating fee rewards after claiming. This implementation prevents an user/attacker from profiting from the fee rewards by continuously buying and selling back-to-back.

Centralization Risks

There are only two important centralization risks in the codebase:

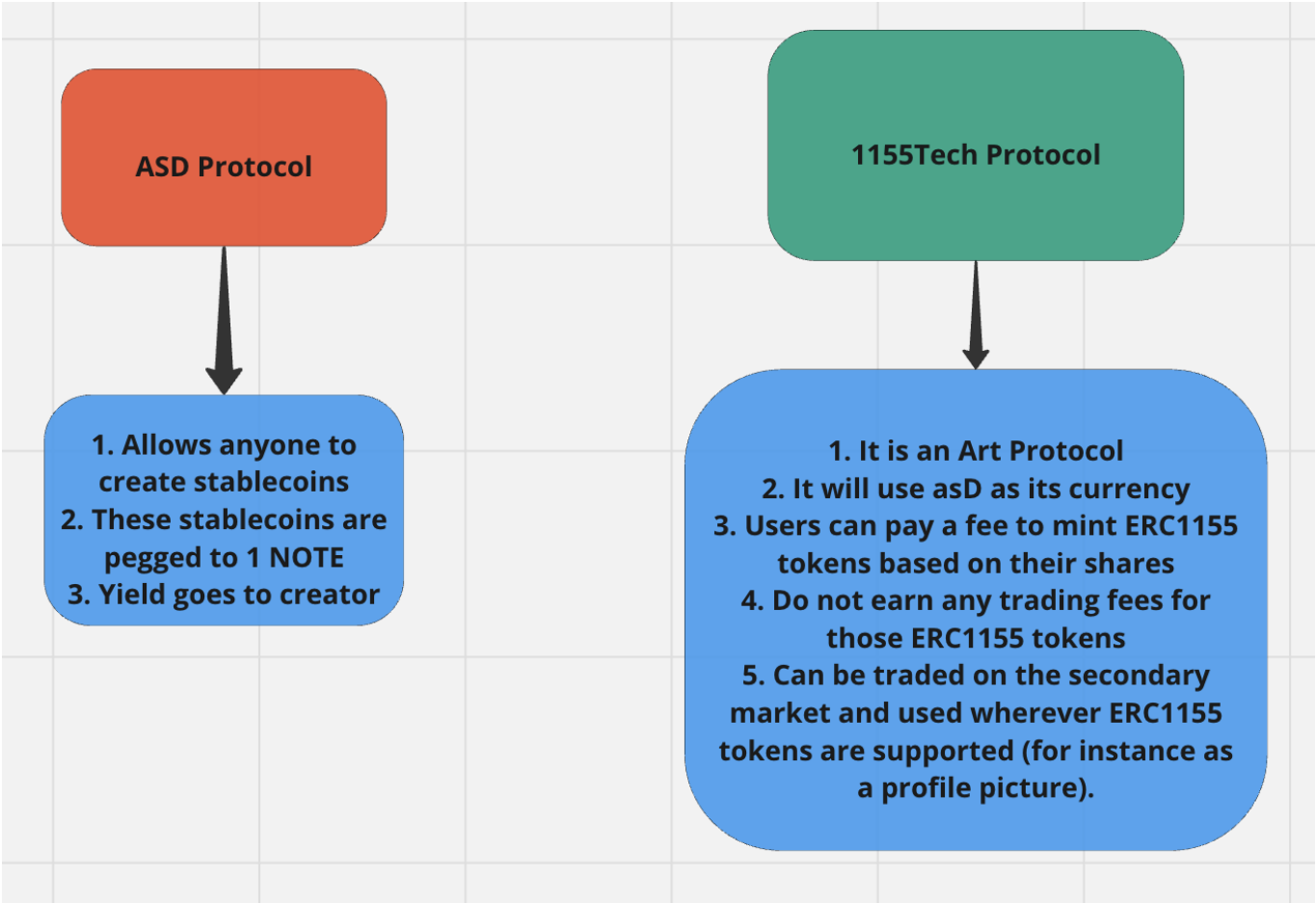
1. The owner of the Market.sol contract can become malicious and restrict creation of new shares and blacklisting creator addresses.
2. The creator of the shareID has access to buying, selling, minting and burning, which allows him to market-make. Another risk that the creator poses is that of deleting the images from the IPFS bucket, rendering the ERC1155 tokens without any image and thus useless if used as a pfp. This can be a loss for the users who bought the NFT since there would be less buyers to buy the NFT now.

Resources used to gain deeper context on the codebase

1. Reading about the CLM
2. Reading about $NOTE
3. Reading about cTokens

Mechanism Review

Features of the asD and 1155Tech protocols

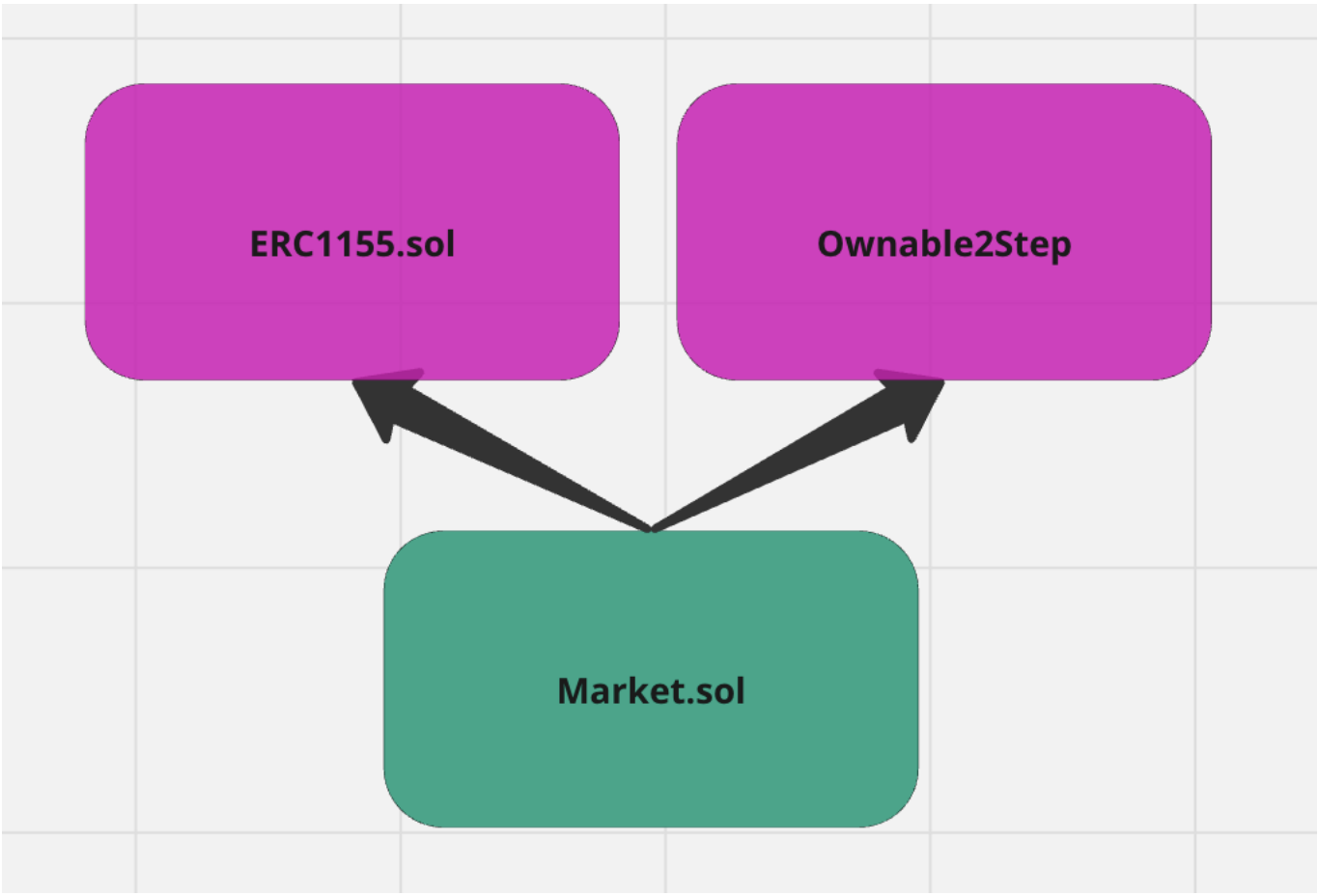


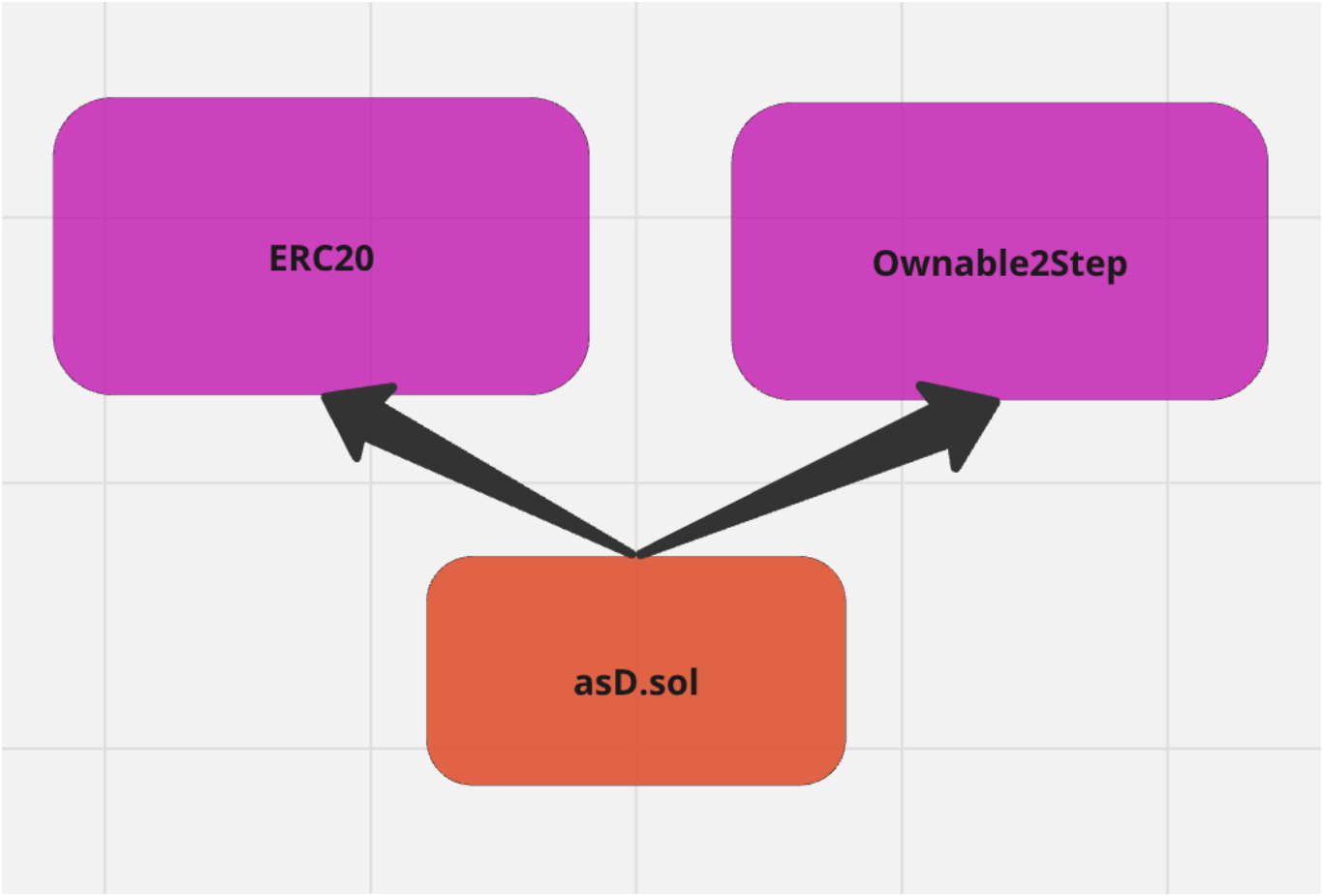
Chains supported

- Application specific dollars will only be deployed on Canto. 1155tech may be deployed on other chains in the future (with a different payment token), but the current focus / plan is also the deployment on Canto

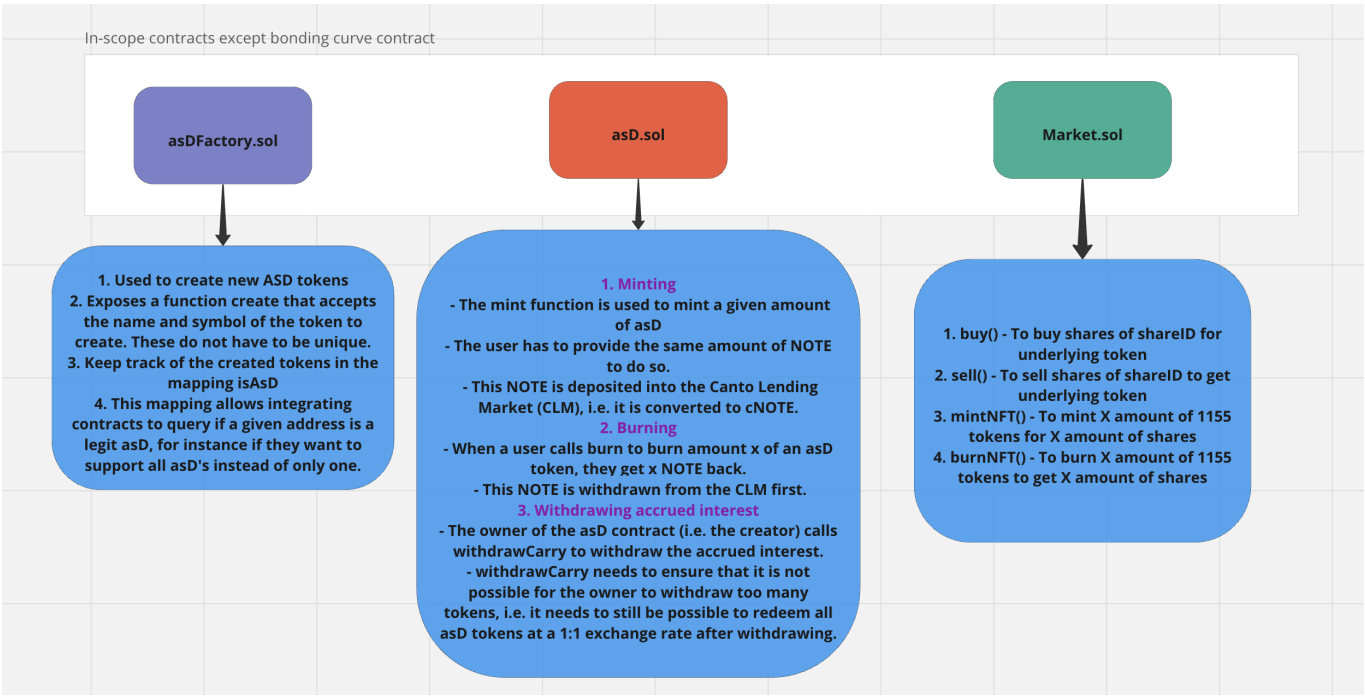
Documentation/Mental models

Inheritance structure





Features of the contracts



Fee model formula

Credits to sponsor for explanation:

Formula: `0.1 / log2(shareCount)`

The intuition behind this is that the fee should decrease for larger markets (hence the division by the share count), but it should only do so slowly / sub-linear (hence the log2).

Systemic Risks/Architecture-level weak spots and how they can be mitigated

1. Preventing centralization - A mutlisig should be used for the deployer of the Market.sol contract since it has the power to block creators from creating new shares as well as steal the platform pool fees
2. Using team's uri as a source of all shareID uris - Currently creators can just delete the media from their URI pointers i.e. IPFS buckets. This is a threat to the users of the 1155Tech protocol since the NFT can lose its value. It would be better to ask the creator to provide the media to the admin team to ensure only the admin have control over setting the media. The admin team can be made into a multisig to ensure no bad actors compromise creator's media.
3. Two shareIDs can point to the same URI, which can be seen as indirect theft of media. There should be some sort of mechanism to prevent setting the same uri for two shareIDS. Even if it is a feature, the creator should be forced to use a different bucket hash for the other shareID. This will ensure no two shareIDs have the same media in use.

FAQ for understanding this codebase

1. What is Turnstile?

- Its a registry system that register any asD token that was created

2. Are there market contracts for every aSD?

- No, there probably will not be a 1:1 mapping between markets and asD tokens. One asD tokens can have none associated markets or it could also have multiple ones (although that is rather unusual)

Time spent:

15 hours