tags: `Final Report`

Fountain Protocol Audit

Feb 18, 2022
by **Verilog Audit**

This report presents Verilog's smart contract auditing engagement with Fountain Protocol. Fountain Protocol is one of the first Lending protocols on the Emerald Paratime of Oasis Network.

Table of Content

- Fountain Protocol Audit
  - Table of Content
  - Project Summary
  - Service Scope
  - Architecture
  - Privileged Roles
  - Findings & Suggestions for Improvement
    - Critical
    - Major
    - Medium

        - Minor
        - Informational
    - Disclaimer

Project Summary

Fountain Protocol is a high capital efficiency, one-stop capital management platform for users' DeFi Assets. Fountain Protocol is able to take advantage of the extremely efficient and low-cost Oasis Network and create a fund pool with a diverse source of revenue and DeFi applications.

Service Scope

The smart contract audit was conducted over 1 week, from Feb 14 to Feb 18, 2022 by the Verilog team. Our audit is conducted on the **main** branch (https://github.com/dev-fountain/fountain-protocol), with commit hash cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/tree/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

Our engagement with Fountain Protocol includes the following two services:

- Pre-Audit Consulting Service
- Audit Service

1. **Pre-Audit Consulting Service**

   As a part of the pre-audit service, the Verilog team worked closely with the Fountain development team to discuss potential vulnerability and smart contract development best practices in a timely fashion. Verilog team is very appreciative for establishing an efficient and effective communication channel with the Fountain team, as new findings were often exchanged promptly and fixes were deployed quickly, during the preliminary report stage.

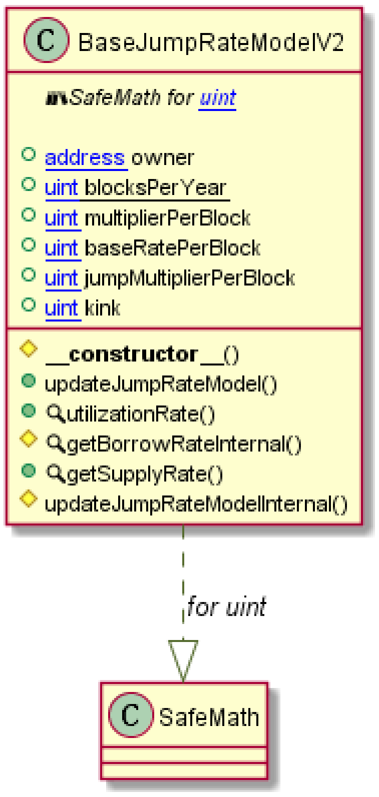
2. **Audit Service**

   The Verilog team conducted a thorough study of the Fountain code, with the Fountain architecture graph and UML graph presented below in the Fountain Architecture section. The list of findings, along with the severity and solution, is available under section **Findings & Improvement Suggestions**.

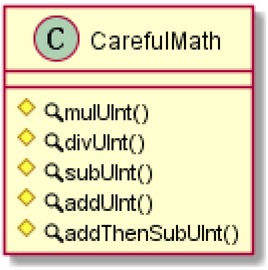
Architecture

These are the major smart contracts in the Fountain Protocol:

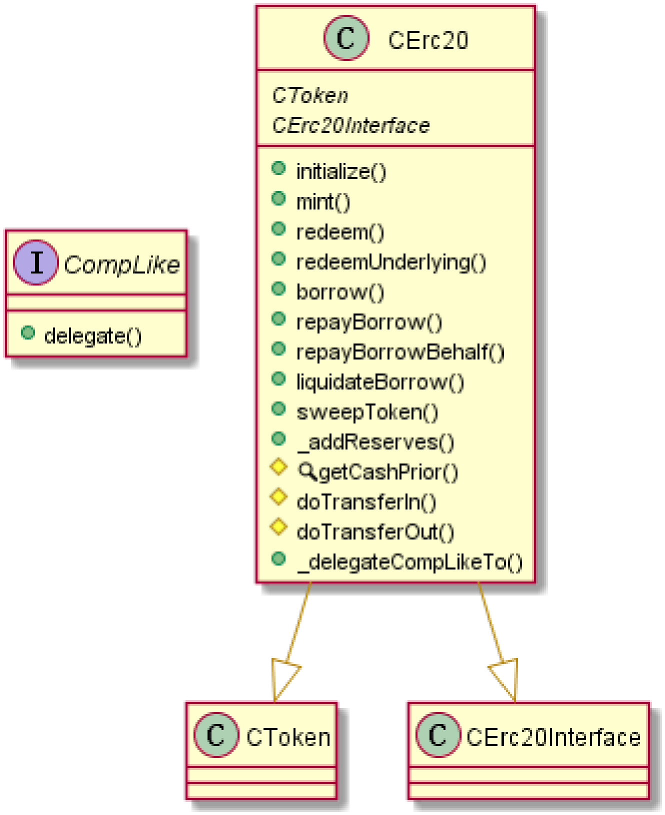
- **dex(folder)**: Uniswap V2 DEX interface
- **Governance(folder):**
  - `Ftp.sol` : Ftp Governance Token Smart Contract
- **interface(folder):**
  - `IComptroller.sol` : compound controller interface
- **Lens(folder):**
  - `CompoundLens.sol`
- **periphery(folder)**
  - `TransferHelper.sol`
  - `LPFarm.sol` : liquidity mining contract
  - `Stake.sol` : single token staking contract
- **`BaseJumpRateModelV2.sol`** : logic for Compound’s Jump Rate Model
- **`CarefulMath.so`** **l**: math library
- **`CErc20.sol`** : Compound’s CErc20 Contract
- **`CEther.sol`** : Compound’s CEther Contract
- **`Comptroller.sol`** : Compound’s Comptroller Contract
- **`CToken.sol`** : Compound’s CToken Contract



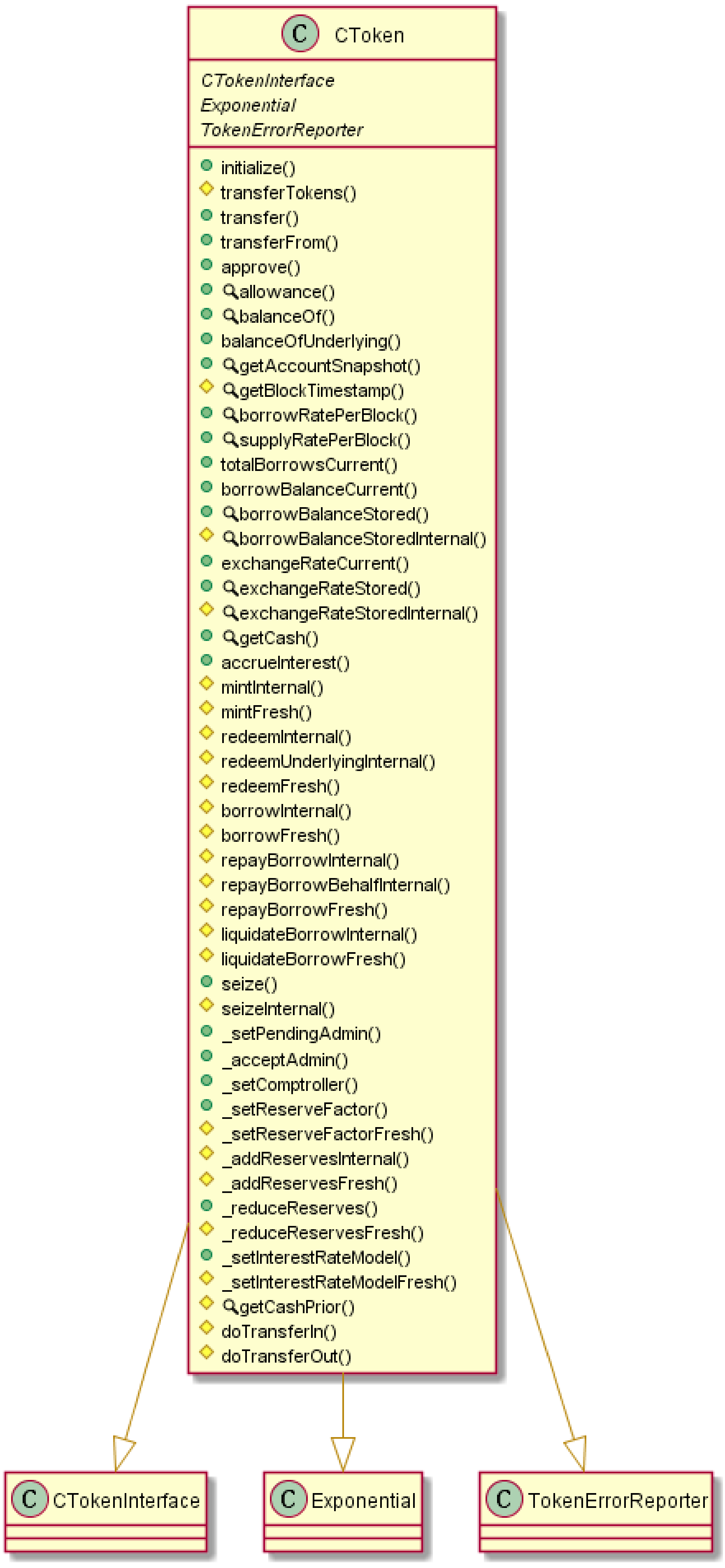
BaseJumpRateModel.sol



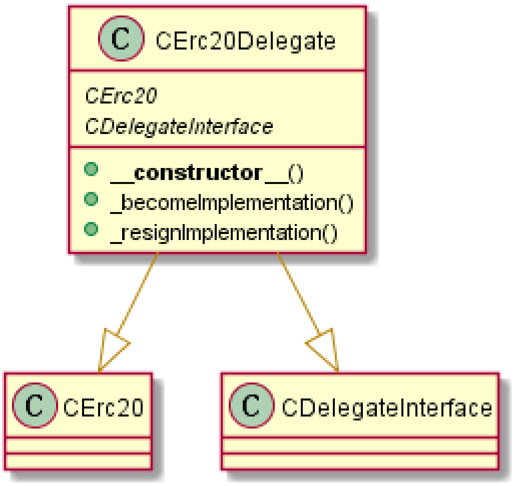
CarefulMath.sol



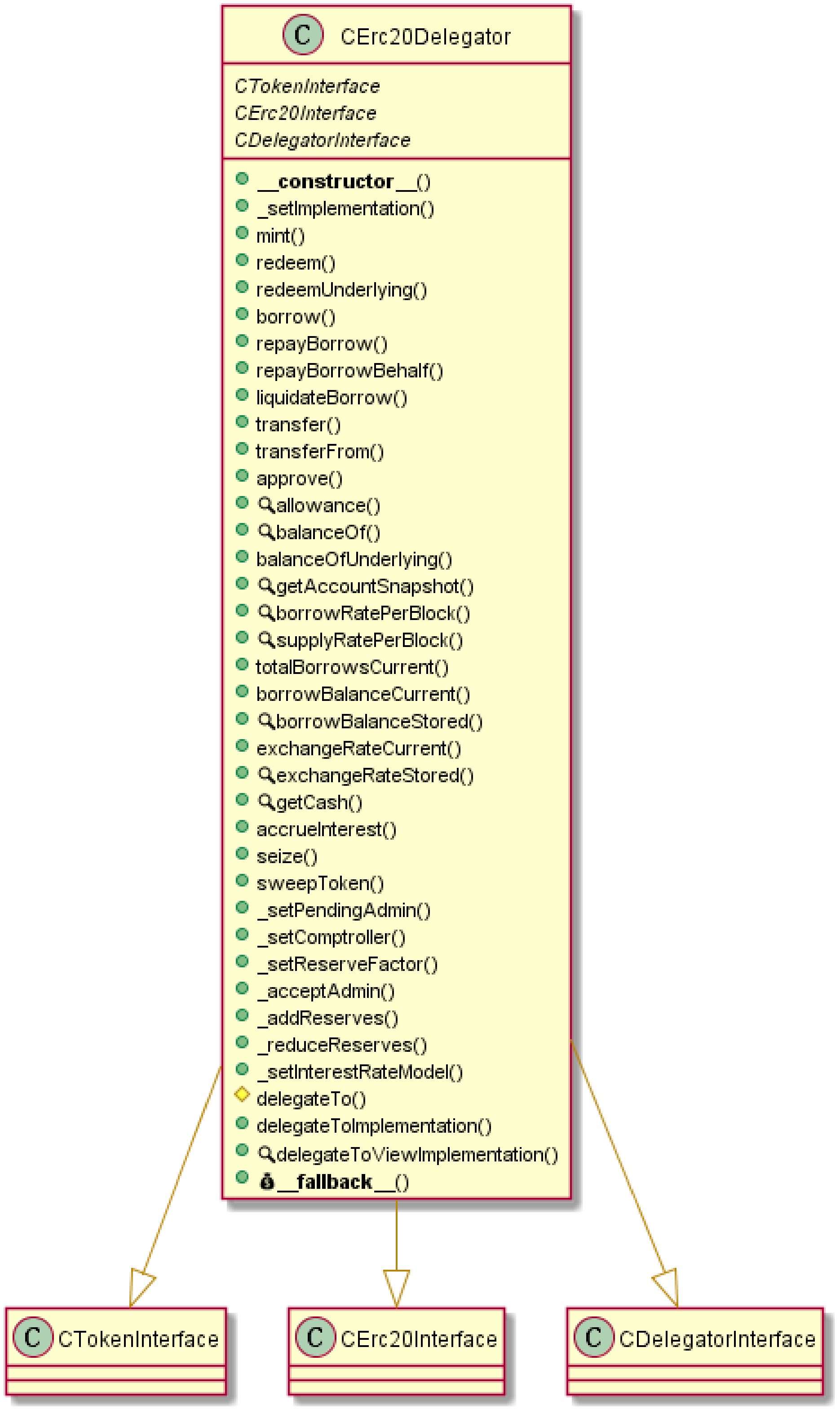
CErc20.sol


CToken
CTokenInterface
Exponential
TokenErrorReporter
initialize()
transferTokens()
transfer()
transferFrom()
approve()
allowance()
balanceOf()
balanceOfUnderlying()
getAccountSnapshot()
getBlockTimestamp()
borrowRatePerBlock()
supplyRatePerBlock()
totalBorrowsCurrent()
borrowBalanceCurrent()
borrowBalanceStored()
borrowBalanceStoredInternal()
exchangeRateCurrent()
exchangeRateStored()
exchangeRateStoredInternal()
getCash()
accrueInterest()
mintInternal()
mintFresh()
redeemInternal()
redeemUnderlyingInternal()
redeemFresh()
borrowInternal()
borrowFresh()
repayBorrowInternal()
repayBorrowBehalfInternal()
repayBorrowFresh()
liquidateBorrowInternal()
liquidateBorrowFresh()
seize()
seizeInternal()
_setPendingAdmin()
_acceptAdmin()
_setComptroller()
_setReserveFactor()
_setReserveFactorFresh()
_addReservesInternal()
_addReservesFresh()
_reduceReserves()
_reduceReservesFresh()
_setInterestRateModel()
_setInterestRateModelFresh()
getCashPrior()
doTransferIn()
doTransferOut()
CTokenInterface
Exponential
TokenErrorReporter

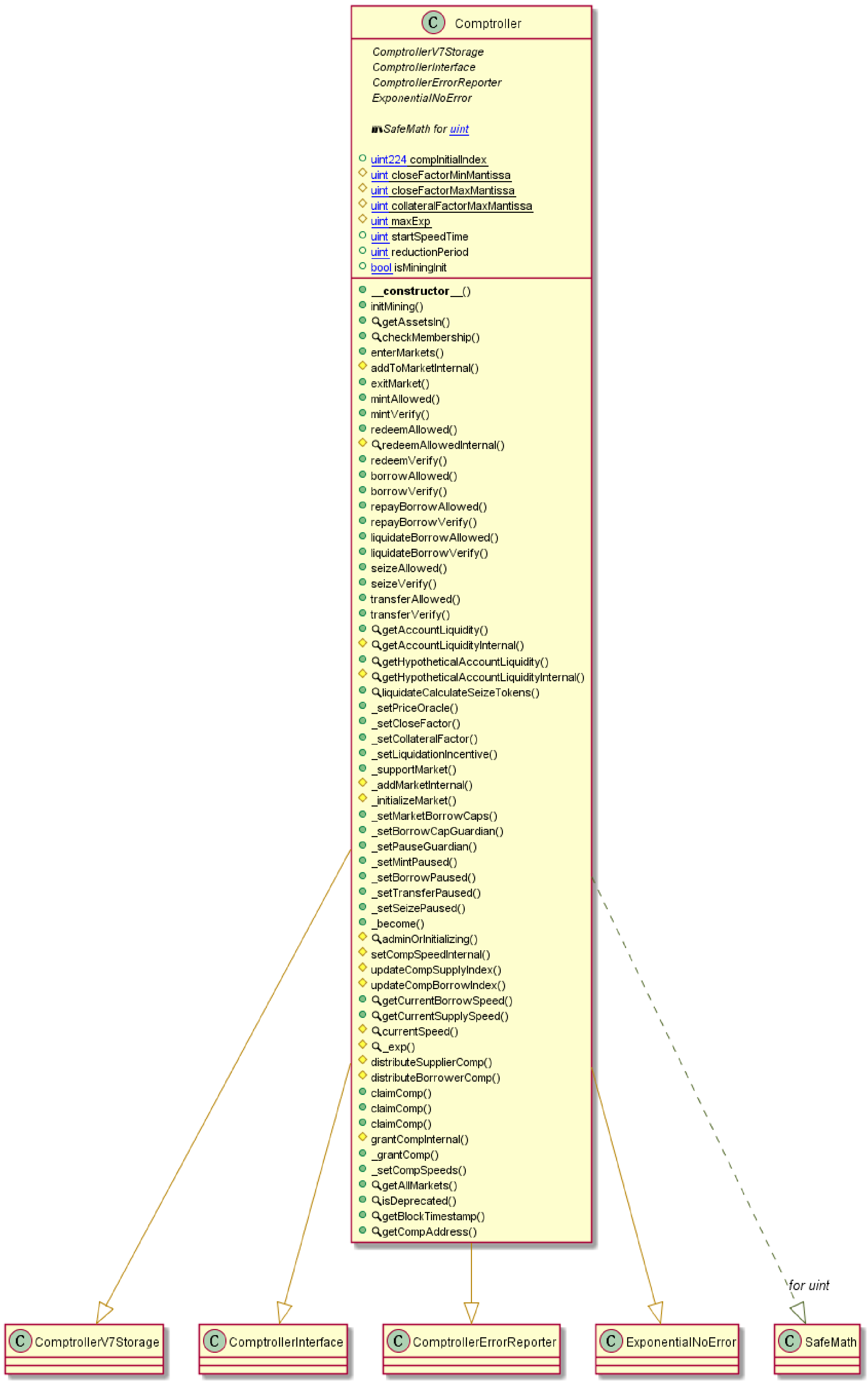
CToken.sol



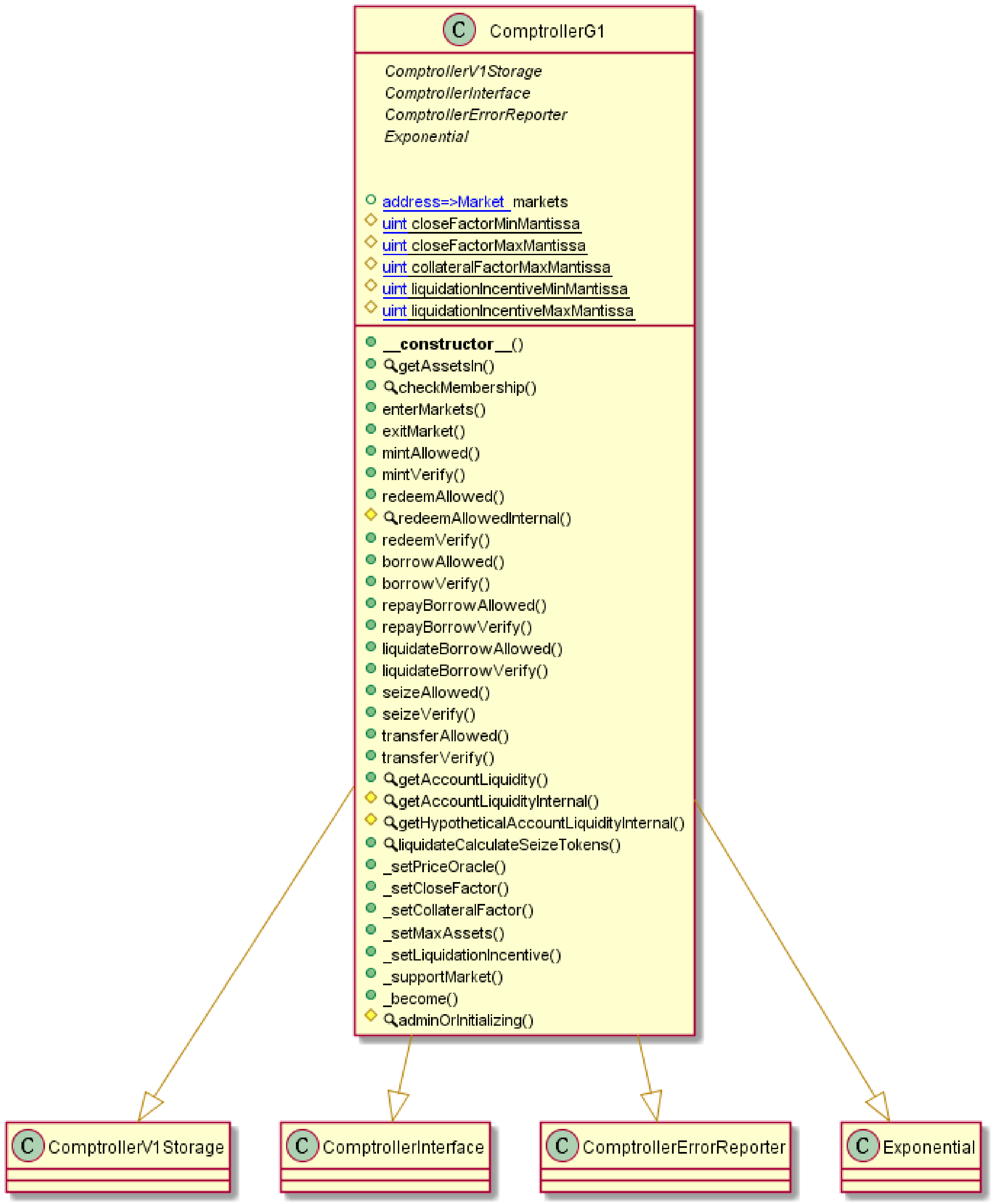
CErc20Delegate.sol



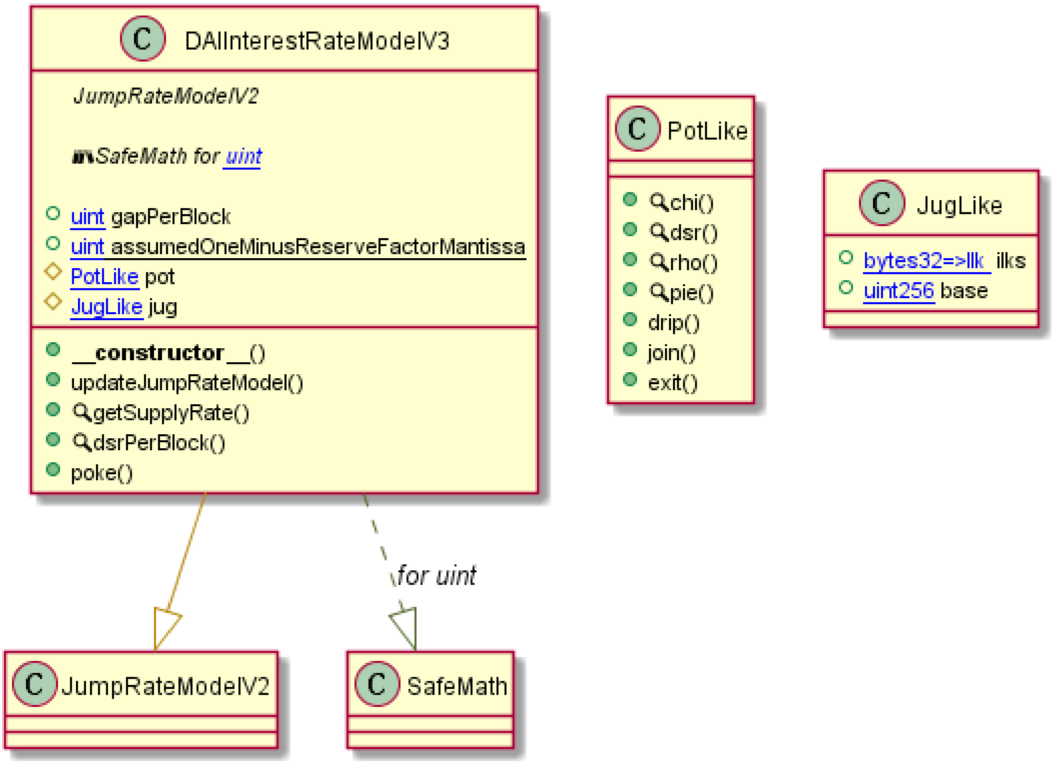
CErc20Delegator.sol



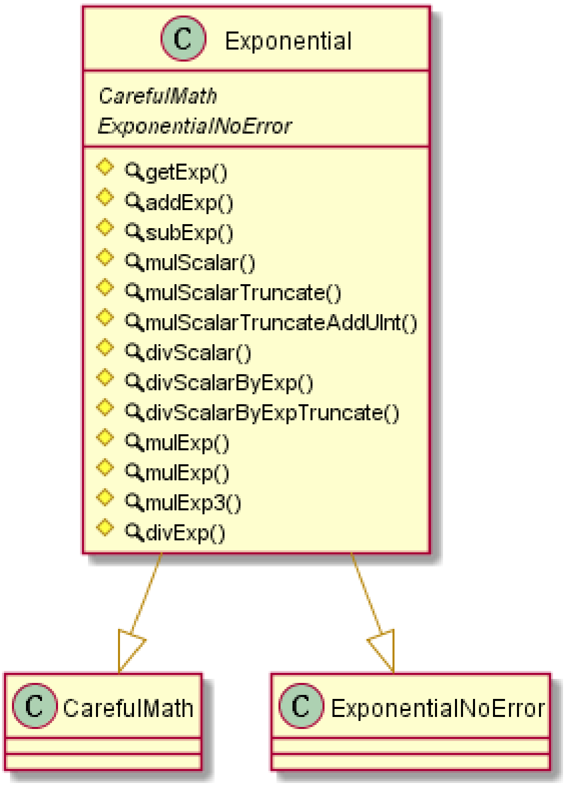
Comptroller.sol



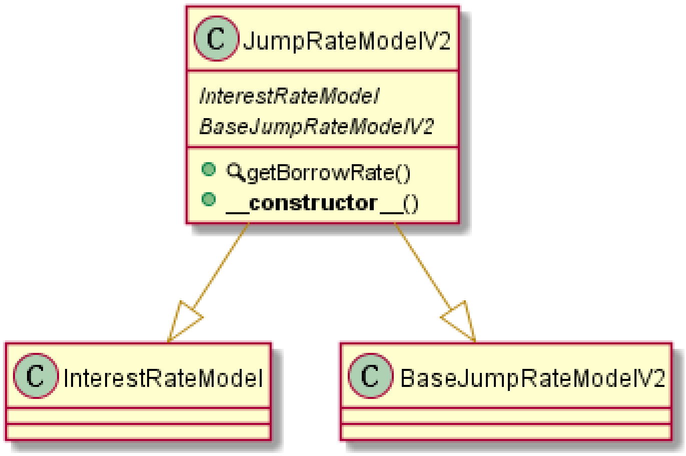
ComptrollerG1.sol



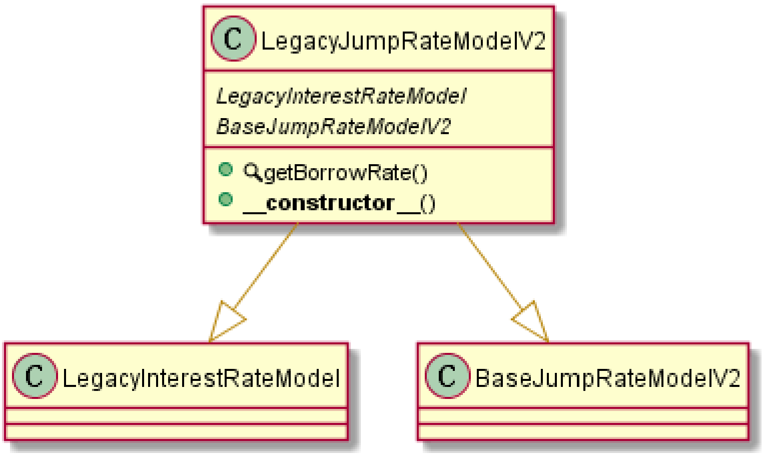
DAIInterestRateModelV3.sol



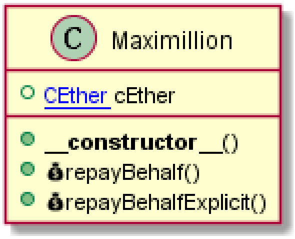
Exponential.sol



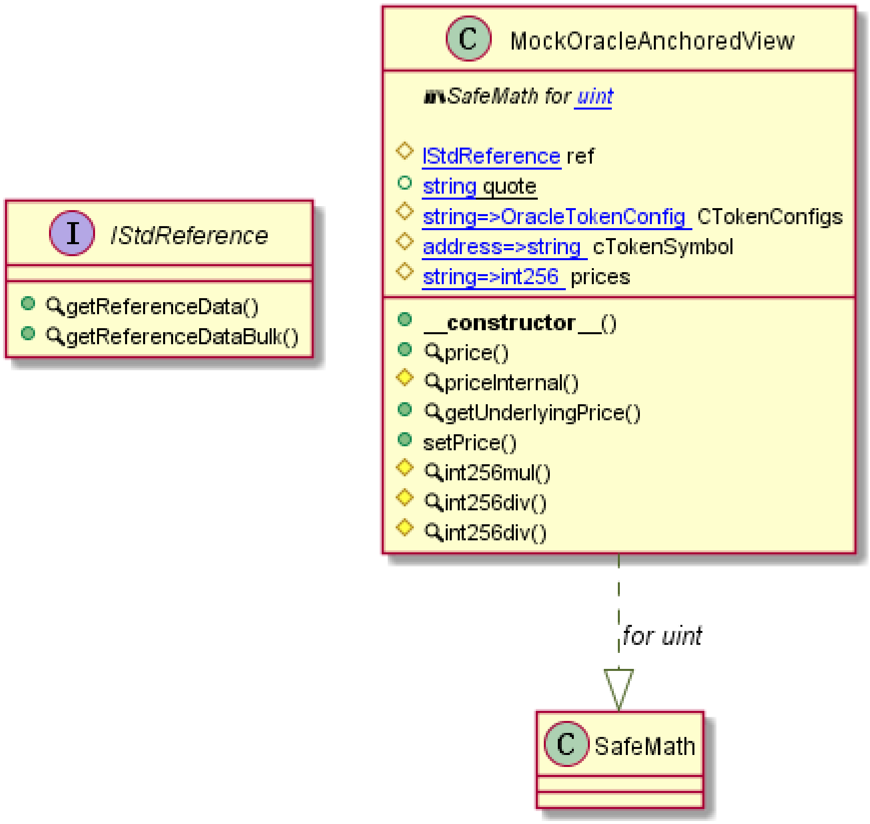
JumpRateModelV2.sol



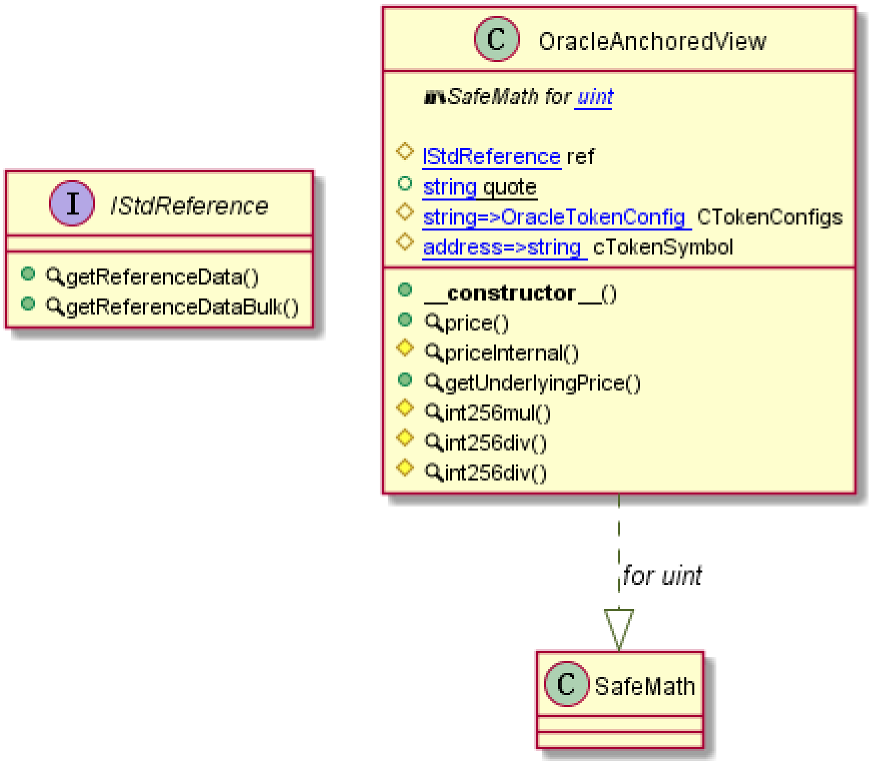
LegacyJumpRateModelV2.sol



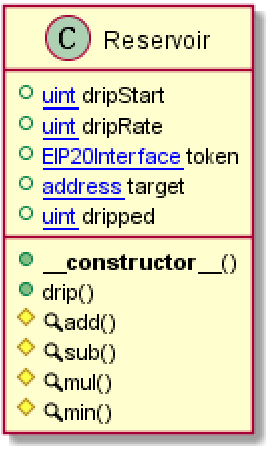
Maximillion.sol



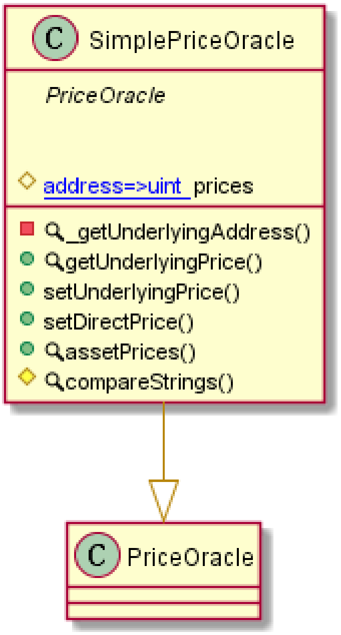
MockOracleAnchoredView.sol



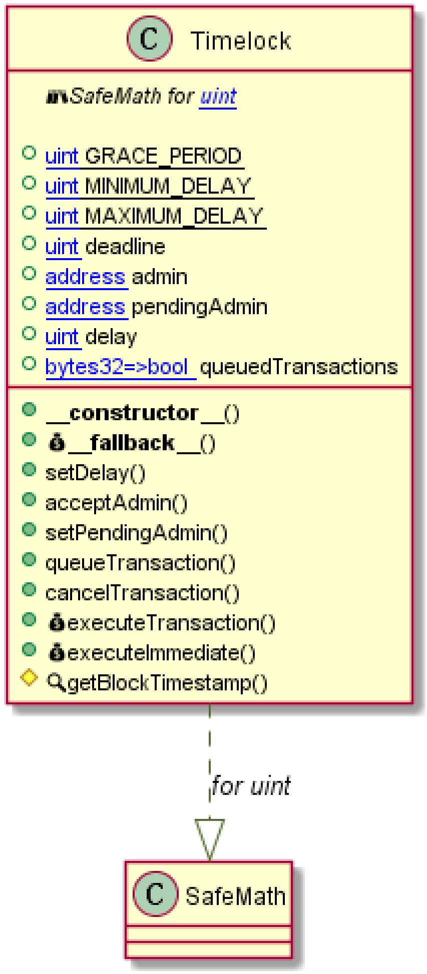
OracleAnchoredView.sol



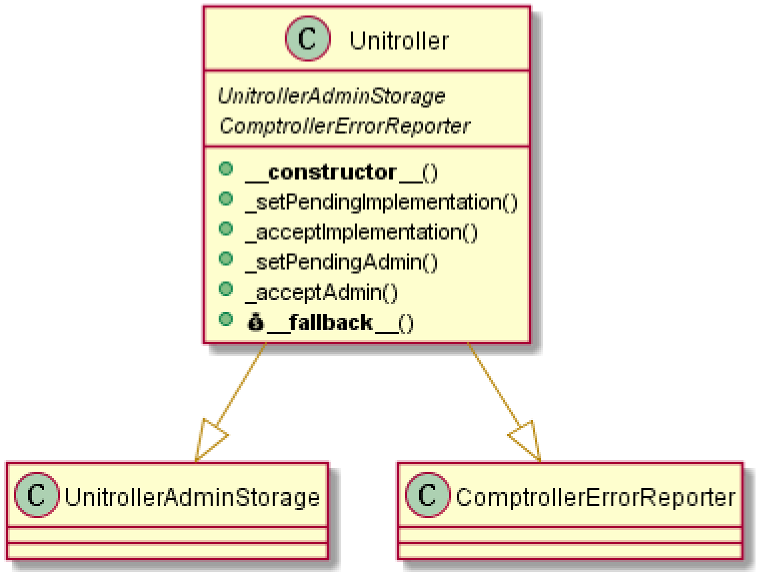
Resevoir.sol



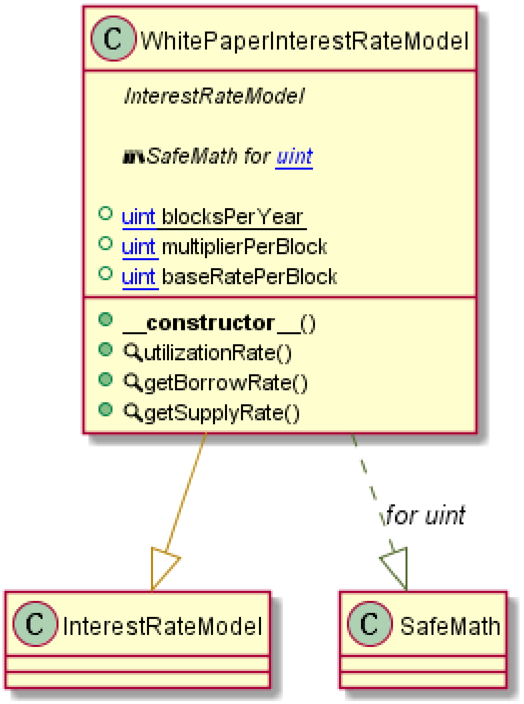
SimplePriceOracle.sol



Timelock.sol



Unitroller.sol



WhitePaperInterestRateModel.sol

Privileged Roles

1. `LPFarm.sol`
   `owner` can updateMultiplier, add, set, setRewardsPerSecond.

2. `Stake.sol`
   `owner` can updateMultiplier, setRewardPerSecond.

3. `BaseJumpRateModelV2.sol`
   `owner` can updateJumpRateModel.

4. `Comptroller.sol`
   `admin` is the deployer of the contract, can initMining, reInitMining, setPriceOracle, setCloseFactor, setCollateralFactor, setLiquidationIncentive, supportMarket, setBorrowCapGuardian, setPauseGuardian, setMintPaused, setBorrowPaused ... etc.

5. `CToken.sol`
   `admin` can initialize CToken Contract.

Findings & Suggestions for Improvement

Informational **Minor** **Medium** **Major** **Critical**

	Total	Acknowledged	Resolved
Critical	1	1	1
Major	0	0	0
Medium	3	3	3
Minor	16	16	10

Critical

1. Use SafeMath Library for arithmetic operations. (`Ftp.sol`) **Critical**
   Description: Arithmetic operations reverting on underflow and overflow is a feature only available in solidity `^0.8.0` .
   Recommendation: Use `SafeMath` library or change solidity version to `^0.8.0` .
   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

Major

none ;)

Medium

1. Call `updatePool()` when updating `BONUS_MULTIPLIER` and `rewardsPerSecond` . (`Stake.sol` , `LPFarm.sol`) **Medium**
   Description: `UpdatePool` needs to be called when updating `BONUS_MULTIPLIER` and `rewardsPerSecond` .
   Recommendation: Pool's reward variables need to be updated. So `UpdatePool` needs to be called when updating `BONUS_MULTIPLIER` and `rewardsPerSecond` .
   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

2. Insufficient reward token balance in pool might cause transaction to fail when users try to claim their accumulated rewards. (`Stake.sol` , `LPFarm.sol`) **Medium**
   Description: The reward token transfer to user will fail if there is no enough reward tokens in pool and user will not be able to `stake` , `withdraw` and `claimReward`
   Recommendation: We recommend to change the `safeRewardsTransfer()` to the followings:

```
function safeRewardsTransfer(address to, uint amount) internal {
     uint rewardTokenBalance = IERC20(rewardsToken).balanceOf(address(this))
     if(amount > rewardTokenBalance) {
         TransferHelper.safeTransfer(rewardsToken, to, rewardTokenBalance);
     } else {
         TransferHelper.safeTransfer(rewardsToken, to, amount);
     }
 }
```

Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

3. As the Solidity version `^0.5.16` is defined in the pragma, the arithmetic calculation should be treated carefully. The function `princeInternal()` and `getUnderlyingPrice()` should be revised by using `SafeMath` library in openzeppelin. (`OracleAnchoredView.priceInternal()` : L52, `OracleAnchoredView.getUnderlyingPrice()` : L59) Medium
   Description: The `int256(data.rate) / 1e10` and `1e28 * rate / config.baseUnit` are not safe.
   Recommendation: Either update the compiler version to be greater than or equal to 0.8.0, or the function should be revised by using `SafeMath` library in openzeppelin.
   Result: Resolved by using `SafeMath` library in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

Minor

1. Require `isMiningInit` to be true in `reInitMining` . (`Comptroller.reInitMining()` : L105) Minor
   Description: Require `isMiningInit` to be true in `reInitMining` .
   Recommendation:

```
function reInitMining(uint256 _reductionPeriod) external{
    require(msg.sender == admin,"only admin can call this function");
    require(isMiningInit, "mining not inited");
    reductionPeriod = _reductionPeriod;
    startSpeedTime = getBlockTimestamp();
}
```

   Result: `reInitMining()` is removed from `Comptroller.sol` in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

2. `nextTotalBorrows` should be less or equal to `borrowCap` (`Comptroller.borrowAllowed()` : L390) **Minor**

   Description: `nextTotalBorrows` should be less or equal to `borrowCap` .
   Recommendation: Replace with the suggested method below.

   ```
   require(nextTotalBorrows <= borrowCap, "market borrow cap reached");
   ```

   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

3. `uint endSpeedTime` depends on literal value (`1440 day`). (`updateCompSupplyIndex()` : L1161, `updateCompBorrowIndex()` : L1199) **Minor**

   Description: `uint endSpeedTime` depends on `1440 day` instead of `uint reductionPeriod` .
   Recommendation: Replace with the suggested method below.

   ```
   uint endSpeedTime = startSpeedTime + reductionPeriod.mul(maxExp);
   ```

   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

4. Missing error message in `getAccountLimits()` (`CompoundLens.sol` : L242). **Minor**

   Description: The `require(errorCode == 0)` statement misses its error message.
   Recommendation: Add error message for the require statement.
   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

5. `lpTokenTotal[pool.lpToken]` is not a secure way of reading lp token amount. (`LPFarm.sol`) **Minor**

   Description: generally speaking, use a mapping to record token in/out amount is not a good practice. A small mistake in withdraw deduction might create flashloan attack oppotunities.
   Recommendation: `pool.lpToken.balanceOf(address(this))` is a more secure way of reading the lp token balance.

   ```
   uint256 lpSupply = pool.lpToken.balanceOf(address(this));
   ```

   Result: This suggestion is not adopted.

6. Based on the above suggestion, `mapping (address => uint) public lpTokenTotal` can be deleted. (`LPFarm.sol`) **Minor**
   Description: if above suggestion been implemented, then there is no need to have the lpTokenTotal mapping.
   Recommendation: using mapping to store lpToken balance is not a secure way of reading balance. Thus, remove this mapping, change to only use IERC20 interface `balanceOf()` is a safer choice.
   Result: This suggestion is not adopted.

7. Maximum Borrow Rate is Too Large for Emerald Chain. The value now is still 0.0005%/block (`CTokenInterface.sol`) **Minor**
   Description: We should decrease this value for Emerald Chain.
   Recommendation: If we have changed the unit of block to second, we should change the value in terms of a time duration.
   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

8. `bool internal _notEntered;` in CTokenInterfaces.sol should be removed and use the nonReentrant instead (`CTokenInterface.sol`) **Minor**
   Description: Since we are using nonReentrant, the mutex variable should be removed.
   Recommendation: Use `nonReentrant` instead.
   Result: This suggestion is not adopted.

9. Consider using Solidity >= 0.8.0 to remove the CarefulMath.sol, for all files using addUInt, subUint, mulUint, and divUint. **Minor**
   Description: After 0.8.0, the overflow and underflow problems are considered by EVM.
   Recommendation: Using pragma solidity ^0.8.0
   Result: This suggestion is not adopted.

10. Should sanitize zero address in `function _setPendingAdmin(address newPendingAdmin) public returns (uint)` . (`Unitroller.sol`) **Minor**
    Description: The `newPendingAdmin` should not be zero address.
    Recommendation: Require `newPendingAdmin` is not zero address.
    Result: This suggestion is not adopted.

11. Should sanitize zero address in `constructor(address admin_, uint delay_)` . (`Timelock.sol`) **Minor**
    Description: The `admin` should not be zero address.
    Recommendation: Require `admin` is not zero address. Also consider passing `msg.sender` to `admin` instead of passing `admin_` .

Result: Zero address check for `admin_` is added, but the input parameter `admin_` instead of suggested `msg.sender` is still assigned to `admin` in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

12. Should sanitize zero address in `constructor(address _owner,address _guardian)` . (`FTPGuardian.sol`) **Minor**
    Description: The `_owner` and `_guardian` should not be zero address.
    Recommendation: Require `_owner` and `_guardian` are not zero address. Also consider passing `msg.sender` to `owner` instead of passing `_owner` .
    Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

Informational

1. Use Openzeppelin Contract Wizards to Generate `Ftp.sol` **Informational**
   Description: current contract code is not updated version of ERC20, although it serves the purposes, but switch to latest version can make project code looks more professional
   Recommendation: generate an ERC20 contract from OpenZeppelin Wizard (https://docs.openzeppelin.com/contracts/4.x/wizard).

   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

2. Move all variables to a storage contract. (`Stake.sol`) **Informational**
   Description: `Stake.sol` is an upgradable contract. Move all variables to a storage contract to prevent storage clashes during future contract upgrades.
   Recommendation: We recommend to move all variables to a storage contract and let the `Stake.sol` inherit the storage contract.

```
 // storge contract
contract StakeStorage {
    address public stakeToken;
    address public rewardsToken;
    uint public rewardsPerSecond;
    uint public BONUS_MULTIPLIER;
    uint public startTime;
    uint public endTime;
    uint public stakeTokenTotal;
    uint public accRewardsPerShare;
    uint public lastRewardBlockTimestamp;
    mapping(address => uint) public userStakeTime;
    mapping(address => uint) public stakeBalance;
    mapping(address => uint) public rewardDebt;
}

// Stake.sol
contract Stake is Initializable, OwnableUpgradeable, ReentrancyGuardUpgradea

}
```

Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

3. Non-constant variable `BONUS_MULTIPLIER` should be in camel case. **Informational**
   Description: Non-constant variable `BONUS_MULTIPLIER` should be camel case.
   Recommendation: Use `bounsMultiplier` to replace `BONUS_MULTIPLIER` .
   Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).

4. Lack of parameter checks (`Stake.initialize()` L30, `Stake.updateMultiplier()` L39, `Stake.setRewardsPerSecond()` L43) **Informational**
   Description: No parameter checks when assigning values to variables.
   Recommendation: Add zero address check for `address` and numbers range check for `uint`
   Result:
   All modifications below are done in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8).
   - Only zero address check is added in `Stake.initialize()` , but not numbers range check.
   - Numbers range check is added in `Stake.updateMultiplier()` .

   - Numbers range check is added in `Stake.setRewardsPerSecond()`

5. Magic number `1e12`. (`Stake.sol` , `LPFarm.sol`) **Informational**
   Description: Magic number `1e12`.
   Recommendation: Make `1e12` a constant variable.
   All modifications above are done in commit [cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)](https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

6. `updateStakingPool()` can be replaced by a single line of code. (`LPFarm.updateStakingPool()` : L87) **Informational**
   Description: There is no need to calculate totalAllocPoint each time adding a pool, simply a line of update in `set()` function.
   Recommendation: Below is the suggested method.

```
 // Update the given pool's reward allocation point. Can only be called by t
function set(
    uint256 _pid,
    uint256 _allocPoint,
    bool _withUpdate
) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
        _allocPoint
    );
    poolInfo[_pid].allocPoint = _allocPoint;
}
```

   Result: This suggestion is not adopted.

7. The address of the reward token should be immutable (`LPFarm.sol` : L22). **Informational**
   Description: The address of the reward token should be immutable.
   Recommendation: Change `address public rewardToken;` to `address immutable public rewardToken;`
   Result: This suggestion is not adopted.

8. The code `totalAllocPoint = totalAllocPoint + _allocPoint;` should be removed from the function `add` (`LPFarm.sol` : L68) **Informational**
   Description: In this function, we call function `updateStakingPool()` in the end, where we recalculate the `totalAllocPoint` .

**Recommendation
**: Remove line 68.
Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

9. Missing `indexed` for important arguments in event `ActionPaused`, `CompGranted`, `CompGranted`, `MarketEntered`, `MarketExited`, and `MarketListed` (`IComptroller`) **Informational**
   Description: The argument `cToken` in event `ActionPaused`, `MarketEntered`, `MarketExited` and `MarketListed`, `recipient` in event `CompGranted` should be indexed.
   Recommendation: The argument `cToken` should be indexed.
   Result: This suggestion is not adopted.

10. Lack of parameter checks (`Comptroller.initMining()` : L97, `Comptroller.reInitMining()` : L105) **Informational**
    Description: No parameter checks when assigning values to variables.
    Recommendation: Add number range check for `_reductionPeriod`
    Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

11. Access control on `msg.sender` address can be abstracted to modifiers. (`Comptroller.initMining()` L97, `Comptroller.reInitMining()` L105, `Comptroller._setPriceOracle()` L845, `Comptroller._setCloseFactor()` L867, `Comptroller._setCollateralFactor` L885, `Comptroller._setPauseGuardian()` L1047) **Informational**
    Description: The repeated uses of checking `msg.sender` can be abstracted to a modifier.
    Recommendation: Replace with the suggested method below.

```
modifier onlyAdmin(){
    require(msg.sender == admin);
    _;
}
```

    Result: This suggestion is not adopted.

12. Constant variables name could be upper case in `Comptroller.sol` . (`compInitialIndex` : L74, `closeFactorMinMantissa` : L77, `closeFactorMaxMantissa` : L80, `collateralFactorMaxMantissa` : L83, `maxExp` : L85) **Informational**
    Description: Constant variables names could be upper case.
    Recommendation: Replace with the variable name with their upper case formats.
    Result: This suggestion is not adopted.

13. Contract `InterestRateModel` should be defined as an interface (Line 7). (`InterestRateModel.sol`) **Informational**
    Description: InterestRateModel should be defined as an interface rather than a contract
    Recommendation: Change it to be an interface.
    Result: This suggestion is not adopted.

14. The redundant `owner` in `event NewGuardian(address owner,address oldGuardian,address newGuardian)` (`FTPGuardian.sol` : L4). **Informational**
    Description: The `owner` is redundant in the event field since `owner` is unchangeable.
    Recommendation: Remove the `address owner` field in the event declaration (L4) and its use (L16).
    Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

15. Nominating the same guardian is allowed in `setGuardian()` (`FTPGuardian.sol` : L12). **Informational**
    Description: The `owner` can set new guardian with the same address as the old guardian.
    Recommendation: Require the `newGuardian` is not equal to the `guardian` .
    Result: Resolved in commit cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8 (https://github.com/dev-fountain/fountain-protocol/commit/cc16318c2db70fdc8fbfb52c26c1f7b9d15875f8)

16. Define FTP addres as a constant or immutable. (`ComptrollerG3.sol` : L1396, `ComptrollerG6.sol` :L1377) **Informational**
    Description: Define FTP addres as a constant or immutable…
    Recommendation: Define FTP addres as a constant or immutable.
    Result: This suggestion is not adopted.

Disclaimer

Verilog receives compensation from one or more clients for performing the smart contract and auditing analysis contained in these reports. The report created is solely for Clients and published with their consent. As such, the scope of our audit is limited to a review of code and only the code we note as being within the scope of our audit detailed in this report. It is important to note that the Solidity code itself presents unique and unquantifiable risks since the Solidity language itself remains under current development and is subject to unknown risks and flaws. Our sole goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies. Thus, Verilog in no way claims any guarantee of security or functionality of the technology we agree to analyze.

In addition, Verilog reports do not provide any indication of the technologies proprietors, business, business model or legal compliance. As such, reports do not provide investment advice and and should not be used to make decisions about investment or involvement with any particular project. Verilog has the right to distribute the Report through other means, including via Verilog publications and other distributions. Verilog makes the reports available to parties other than the Clients (i.e., “third parties”) – on its website in hopes that it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovations.