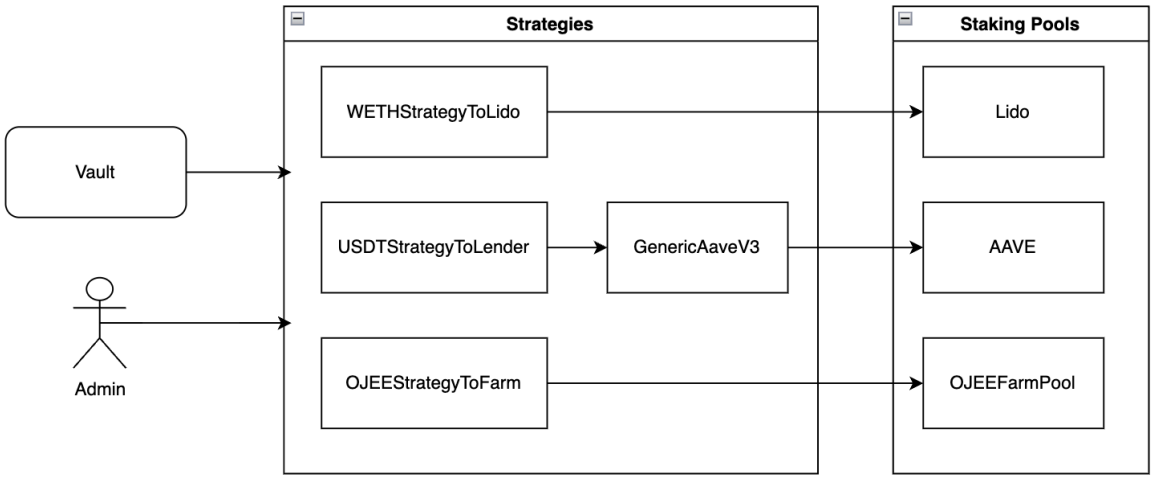
VIMworld Token And Strategy Contracts Techspec

Project Overview

This project offers users a financial management platform where they can invest in various types of tokens such as ETH, USDT, and OJEE. The platform is responsible for interfacing with three staking pools: Lido, AAVE, and OJEEFarmPool. It stakes users' investments in the corresponding staking pools, helping users earn financial returns in ETH, USDT, or OJEE.

Below is the structural diagram of the core components of the project:



Strategies is a core component of this project's contracts, responsible for investing tokens into third-party Staking Pools and retrieving token profits from these Staking Pools.

Vault is the upstream system of Strategy, responsible for interacting with Strategy for investment and profit transactions.

Staking Pools are the downstream system of Strategy, responsible for receiving investments from Strategy and generating profits.

Admin is the administrator, responsible for configuring Strategy, settling profits, etc.

This project has issued ERC20 Tokens on Ethereum: OJEE and POWA. OJEEFarmPool is the platform's self-built OJEE staking pool. In addition to staking profits, the platform will also reward users with extra OJEE and POWA tokens.

1. Functional Requirements

1.1. Roles

Project has three roles:

- Vault: An upstream system of Strategy, it acts like a capital reserve pool or an investment proxy for users. It is responsible for accepting user investments, regularly allocating them to Strategy, and also supporting user withdrawals.
- Admin: Controls the contracts related to Strategy. In this project, Admin is further divided into the following categories:
    - Governance: Has the highest authority and can call all admin-class functions.
    - Management: Ordinary administrators (more upstream management), capable of configuring Strategy parameters, setting whitelists, etc.
    - Guardian: Operations and maintenance monitor, able to temporarily suspend the Strategy contract in emergencies.
    - Strategist: Ordinary administrators (more downstream management), capable of making settings related to the staking pool.
    - Keeper: Settlement officer, able to trigger settlement operations.
    - Owner of the Powa contract: Responsible for adding or removing contract addresses that can mint Powa Tokens.

1.2. Features

The project has the following features:

Strategy:

- Performing harvest settlements, withdrawing profits from the staking pool, and simultaneously reinvesting the funds from the Vault back into the staking pool. (Admin)

- Withdrawing funds from the staking pool. (Vault)
- Transfers all `Vault` from this strategy to new. (Vault)
- Activates emergency exit. Once activated, the Strategy will exit its position upon the next harvest, depositing all funds into the Vault as quickly as is reasonable given on-chain conditions.(Admin)

OJEE:

- Mint 100 billion when it is deployed.
- Burn your own OJEE token. (Everyone)
- Transfer your own OJEE token. (Everyone)

Powa:

- Mint POWA token. (Contract specified by owner)
- Burn your own POWA token. (Everyone)
- Transfer your own POWA token. (Everyone)

1.3. Use Cases

1. When the funds temporarily stored in the Vault by users reach certain conditions, Admin will call Strategy's Harvest for settlement. The settlement involves withdrawing profits from the staking pool connected to the strategy, and investing the funds from the Vault into the staking pool. After the Harvest, users can see an increase in their profits.
2. When users withdraw their principal or profits, if the funds in the Vault are insufficient, the Vault will withdraw the shortfall from the strategy, which in turn withdraws the needed funds from the corresponding staking pool.
3. The Vault calls the migrate function of the strategy to switch to a new strategy, thereby ultimately interfacing with a new staking pool.
4. Admin activates emergency exit. Once activated, the Strategy will exit its position upon the next harvest, depositing all funds into the Vault as quickly as is reasonable given on-chain conditions.
5. User transfer OJEE or POWA token.
6. When users complete certain tasks, the platform rewards them with POWA tokens.

2. Technical Requirements

Project technical specifications

This project has been developed with **`Solidity`** language, using **`Hardhat`** as a development environment. **`Typescript`** is the selected language for testing and scripting.

In addition, **`OpenZeppelin`**'s libraries are used in the project. All information about the contracts library and how to install it can be found in their GitHub(contracts and contracts-upgradeable).

In the project folder, the following structure is found:

```
├── contracts
│   ├── common
│   │   └── Timelock.sol
│   ├── interfaces
│   │   └── IVault.sol
│   ├── mocks
│   │   ├── AdminHelperUpgradeable.sol
│   │   ├── IShareERC20.sol
│   │   ├── MockCommonHealthCheck.sol
│   │   ├── MockERC20TokenFarmPool.sol
│   │   ├── MockProxyAdmin.sol
│   │   ├── MockUSDTToEthOracle.sol
│   │   ├── MockUpgradeableProxy.sol
│   │   ├── MockVault.sol
│   │   ├── ShareERC20.sol
│   │   ├── TestAave.sol
│   │   ├── TestCurveFi.sol
│   │   ├── TestGenericAaveV3.sol
│   │   ├── TestLido.sol
│   │   ├── TestOJEEStrategyToFarm.sol
│   │   ├── TestProtocolDataProvider.sol
│   │   ├── TestStrategy.sol
│   │   ├── TestToken.sol
│   │   ├── TestUSDTStrategyToLender.sol
│   │   ├── TestUniswap.sol
│   │   ├── TestWETHStrategyToLido.sol
```

```
│   │   ├── USDT.sol
│   │   ├── USDTERC20.sol
│   │   └── WETH.sol
│   ├── staking
│   │   ├── ETHStrategies
│   │   │   └── WETHStrategyToLido.sol
│   │   ├── OJEEStrategies
│   │   │   └── OJEEStrategyToFarm.sol
│   │   ├── USDTStrategies
│   │   │   ├── GenericAaveV3.sol
│   │   │   └── USDTStrategyToLender.sol
│   │   ├── base
│   │   │   ├── BaseStrategy.sol
│   │   │   └── GenericLenderBase.sol
│   │   └── interfaces
│   │       ├── IBaseStrategy.sol
│   │       ├── ICurveFi.sol
│   │       ├── IERC20TokenFarmPool.sol
│   │       ├── IGenericLender.sol
│   │       ├── IHealthCheck.sol
│   │       ├── ISteth.sol
│   │       ├── IWETH.sol
│   │       ├── IWantToEth.sol
│   │       ├── aave
│   │       │   ├── DataTypesV3.sol
│   │       │   ├── IAtoken.sol
│   │       │   ├── IPool.sol
│   │       │   ├── IPoolAddressesProvider.sol
│   │       │   ├── IProtocolDataProvider.sol
│   │       │   └── IReserveInterestRateStrategy.sol
│   │       └── uniswap
│   │           └── IUniswapV2Router.sol
│   └── tokens
│       ├── OJEE.sol
│       └── POWA.sol
├── docs
│   └── Requirements.doc
├── README.md
├── hardhat.config.ts
├── package-lock.json
├── package.json
├── scripts
│   ├── deploy.ts
│   └── utils.ts
```

```
├── test
│   ├── staking
│   │   ├── BaseStrategy.test.ts
│   │   ├── GenericAaveV3.test.ts
│   │   ├── OJEEStrategyToFarm.test.ts
│   │   ├── USDTStrategyToLender.test.ts
│   │   └── WETHStrategyToLido.test.ts
│   ├── tokens
│   │   ├── OJEE.test.ts
│   │   └── POWA.test.ts
│   ├── contractHelpers.ts
│   ├── contractsConfig.ts
│   └── utils.ts
└── tsconfig.json
```

Development environment

Start with **`README.md`** to find all basic information about project structure and scripts that are required to test and deploy the contracts.

Inside the **`./contracts`** folder, **`./tokens/*.sol`** contains the smart contract with the explained in section 2.2 of this document.
Inside the **`./contracts/staking`** folder, **`./base/*.sol`**、**`./ETHStrategies/*.sol`**、**`./OJEEStrategies/*.sol`** and **`./USDTStrategies/*.sol`** contains the smart contract with the explained in section 2.2 of this document.

Inside the **`./contracts/staking/base`** folder, **`BaseStrategy`** and **`GenericLenderBase`** import the interface **`IVault`**, from the **`interfaces`** folder that provides the necessary **`Vault`** methods to run the contract.

`BaseStrategy` and **`GenericLenderBase`** are abstract contracts.
`WETHStrategyToLido`、**`USDTStrategyToLender`** and **`OJEEStrategyToFarm`** inherit from **`BaseStrategy`**, while **`GenericAaveV3`** inherits from **`GenericLenderBase`**.

The additional solidity files **`./mocks/*.sol`** are provided in **`mocks`** in order to run the tests for the project.

In the **`./tests`** folder, ***`staking/*.ts and tokens/*.ts`*** provides the tests of the different methods of the main contract in Typescript. In **`contractsConfig.ts`**、**`contractHelpers.ts`** and **`utils.ts`** are test helpers.

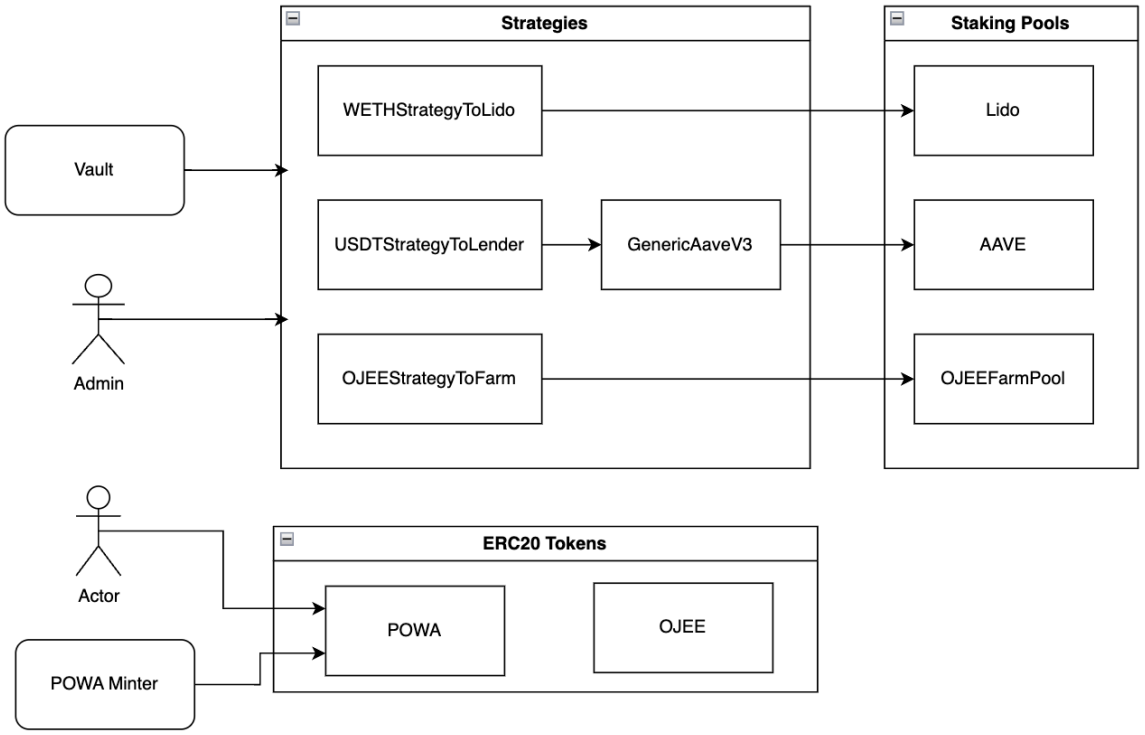
The main contract can be deployed using the **`deploy.ts`** script in **`./deploy`**. In order to do so, **`.env.example`** must be renamed **`.env`**, and all required data must be provided.

The project configuration is found in **`hardhat.config.ts`**, where dependencies are indicated. Mind the relationship of this file with **`.env`**. A basic configuration of Polygon's Mumbai testnet is described in order to deploy the contract. And an etherscan key is set to to configure its different functionalities directly from the repo. More information about this file's configuration can be found in the Hardhat Documentation.

Finally, this document can be found in **`./docs`**.

2.1. Architecture Overview

The following chart provides a general view of project contract Hierarchy and structure.



2.2. Contract Information

This section contains detailed information (their purpose, assets, functions, and events) about the contracts used in the project.

2.2.1. OJEE.sol

A standard ERC20 token contract. Users have the ability to burn OJEE tokens in their wallets at their discretion. OJEE are initially minted to one hundred billion when contract deploymenting, and no further minting is permitted after deployment.

The following description will introduce the differences compared to ERC20 contracts.

`Contract deployment`

When the contract is deployed, 100 billion OJEE will be initially minted into the designated wallet. It cannot be minted later.

2.2.1.1. Functions

OJEE has the following functions:

- **`burn:`** Burn the specified amount of OJEE in the caller's wallet.

2.2.2. POWA.sol

A ERC20 token contract. The contract inherits from Openzeppelin's ERC20Burnable and Ownable. Users have the ability to burn POWA in their wallets at their discretion, and also can approve allowance to burn user POWA to other addresses. POWA can be minted later by the admin.

The following description will introduce the differences compared to ERC20Burnable contracts.

`Contract deployment and mint POWA`

POWA will not be minted when the contract is deployed, but will be minted later according to project rules.

`Setting permissions`

Grant the specified contract permission to mint POWA.

`Getter functions`

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract:

- **`minters()`** → returns whether the address has the mint permission of POWA.

2.2.2.1. Assets

POWA contains the following entities:

- **`minters`**: mapping that store whether each wallet address has mint permissions.

2.2.2.2. Events

POWA has the following events:

- **`EventSetMinter:`** Emitted when setting minting permission for the address with POWA, the parameter represents the wallet address obtaining minting privileges.
- **`EventUnsetMinter:`** Emitted when revoking minting permission for the address with POWA, the parameter represents the wallet address for which minting privileges are being revoked.

2.2.2.3. Modifiers

POWA has the following modifiers:

- **`onlyMinter:`** Checks if msg.sender has minting privileges for POWA.

2.2.2.4. Functions

POWA has the following functions:

- **`mint:`** Mint a specified amount of POWA to the designated wallet address.
- **`setMinter:`** Grant minting permission for POWA to the wallet address.
- **`unsetMinter:`** Revoke minting permission for POWA from the wallet address.
- **`minters:`** Returns whether the address has the mint permission of POWA.

2.2.3. BaseStrategy.sol

An abstract upgradeable contract that implements all of the required functionality to interoperate closely with the Vault contract. This contract should be inherited, and the abstract methods should be implemented to adapt the Strategy to the particular needs it has to create a return.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract, their explanation can be found in the Functions header:

- **apiVersion()**
- **name()**
- **delegatedAssets()**
- **ethToWant()**
- **estimatedTotalAssets()**
- **isActive()**
- **tendTrigger()**
- **harvestTrigger()**

2.2.3.1. Assets

The contract contains the following entities:

- **vault:** The address of the upstream “Vault” contract to be connected.
- **strategist:** The address of the “strategist” role.
- **keeper:** The address of the “keeper” role.
- **want:** ERC20 token address of the invest token.
- **emergencyExit:** Mark whether the contract is in an emergency exit state.
- **doHealthCheck:** Mark whether the contract needs to perform health checks when calling “harvest()”.
- **healthCheck:** The address of the customized health check contract.
- **metadataURI:** String that stores the URI of the file describing the strategy.
- **minReportDelay:** The minimum number of seconds between harvest calls, used to determine whether to recommend triggering “harvest()”.

- **maxReportDelay:** The maximum number of seconds between harvest calls, used to determine whether to recommend triggering “harvest()”.
- **profitFactor:** The minimum multiple that `callCost` must be above the credit/profit to be "justifiable", used to determine whether to recommend triggering “harvest()”.
- **debtThreshold:** Use this to adjust the threshold at which running a debt causes a harvest trigger, used to determine whether to recommend triggering “harvest()”.

2.2.3.2. Events

The contract has the following events:

- **Harvested:** Emitted when the address with keeper permissions calls “harvest()”. The argument profit represents the profit earned between the last harvest and this harvest, loss represents the loss that occurred between the last harvest and this harvest, debtPayment represents the amount of debt actually returned to the vault contract by this harvest, debtOutstanding represents the amount of debt owed to the vault contract that should be repaid by this harvest.
- **UpdatedStrategist:** Emitted when setting up a new strategist role. The argument newStrategist represents the address of the new strategist role.
- **UpdatedKeeper:** Emitted when setting up a new keeper role. The argument newKeeper represents the address of the new keeper role.
- **UpdatedMinReportDelay:** Emitted when setting the new minReportDelay. The argument delay represents the new minReportDelay value.
- **UpdatedMaxReportDelay:** Emitted when setting the new maxReportDelay. The argument delay represents the new maxReportDelay value.
- **UpdatedProfitFactor:** Emitted when setting the new profitFactor. The argument profitFactor represents the new profitFactor value.

- **UpdatedDebtThreshold:** Emitted when setting the new debtThreshold. The argument debtThreshold represents the new debtThreshold value.
- **UpdatedMetadataURI:** Emitted when setting the new metadataURI. The argument uri represents the new metadataURI value.
- **SetHealthCheck:** Emitted when setting up the Healthcheck contract. The argument metadataURI represents the address of the HealthCheck contract.
- **SetDoHealthCheck:** Emitted when setting whether to enable health check. The argument healthCheck represents whether to enable it.
- **EmergencyExitEnabled:** Emitted when the strategy is exited urgently.

2.2.3.3. Modifiers

The contract has the following modifiers:

- **onlyInitializing:** Checks if the contract is initializing.
- **onlyAuthorized:** Checks if msg.sender has this permission. This permission includes these roles: strategist, vault's governance.
- **onlyEmergencyAuthorized:** Checks if msg.sender has this permission. This permission includes these roles: strategist, vault's governance, vault's guardian, vault's management.
- **onlyStrategist:** Checks if msg.sender has this permission. This permission includes these roles: strategist.
- **onlyGovernance:** Checks if msg.sender has this permission. This permission includes these roles: vault's governance.
- **onlyKeepers:** Checks if msg.sender has this permission. This permission includes these roles: keeper, strategist, vault's governance, vault's guardian, vault's management.
- **onlyVaultManagers:** Checks if msg.sender has this permission. This permission includes these roles: vault's management, vault's governance.

2.2.3.4. Functions

The contract has the following functions:

- __BaseStrategy_init: internal method that initializes the Strategy. This is called only once when the contract is deployed. Call method __BaseStrategy_init_unchained().
- __BaseStrategy_init_unchained: internal method that initializes the Strategy's personalized configuration. This is called only once when the contract is deployed. Sets these variables: vault, want, strategist, keeper, minReportDelay, maxReportDelay, profitFactor, debtThreshold.
- tend: adjust the Strategy's position. The purpose of tending isn't to realize gains, but to maximize yield by reinvesting any returns. "onlyKeepers" can call this function.
- harvest: harvests the Strategy, recognizing any profits or losses and adjusting the Strategy's position.In the rare case the Strategy is in emergency shutdown, this will exit the Strategy's position. "onlyKeepers" can call this function.
- withdraw: withdraws `amountNeeded_` to `vault`. Only `Vault` can call this function.
- migrate: transfers all `want` from this Strategy to `newStrategy_`. Only `Vault` can call this function.
- setEmergencyExit: activates emergency exit. Once activated, the Strategy will exit its position upon the next harvest, depositing all funds into the Vault as quickly as is reasonable given on-chain conditions. "onlyEmergencyAuthorized" can call this function.
- sweep: removes tokens from this Strategy that are not the type of tokens managed by this Strategy. This may be used in case of accidentally sending the wrong kind of token to this Strategy. "onlyGovernance" can call this function.
- setHealthCheck: sets up the address of the HealthCheck contract. "onlyVaultManagers" can call this function.
- setDoHealthCheck: sets up whether to enable it. "onlyVaultManagers" can call this function.

- setStrategist: sets up the address of the new strategist role. "onlyAuthorized" can call this function.
- setKeeper: sets up the address of the new keeper role. "onlyAuthorized" can call this function.
- setMinReportDelay: sets up the new minReportDelay value. "onlyAuthorized" can call this function.
- setMaxReportDelay: sets up the new maxReportDelay value. "onlyAuthorized" can call this function.
- setProfitFactor: sets up the new profitFactor value. "onlyAuthorized" can call this function.
- setDebtThreshold: sets up the new debtThreshold value. "onlyAuthorized" can call this function.
- setMetadataURI: sets up the new metadataURI value. "onlyAuthorized" can call this function.
- apiVersion: returns version of this Strategy implements.
- isActive: returns whether the strategy is actively managing a position.
- name: returns this Strategy's name. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- delegatedAssets: returns the amount (priced in want) of the total assets managed by this strategy should not count towards VIMWorld's TVL calculations. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- ethToWant: returns an accurate conversion from `amtInWei_` to `want` . This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- estimatedTotalAssets: returns an accurate estimate for the total amount of assets (principle + return) that this Strategy is currently managing, denominated in terms of `want` tokens. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.

- tendTrigger: returns whether `tend()` should be called. The keeper will provide the estimated gas cost that they would pay to call `tend()`, and this function should use that estimate to make a determination if calling it is "worth it" for the keeper.This function can be overridden to adapt the strategy to its specific needs.
- harvestTrigger: returns whether `harvest()` should be called. The keeper will provide the estimated gas cost that they would pay to call `harvest()`, and this function should use that estimate to make a determination if calling it is "worth it" for the keeper.This function can be overridden to adapt the strategy to its specific needs.
- _prepareReturn: internal method that performs any Strategy unwinding or other calls necessary to capture the "free return" this Strategy has generated since the last time its core position(s) were adjusted. This call is only used during "normal operation" of a Strategy, and should be optimized to minimize losses as much as possible. This method returns any realized profits and/or realized losses incurred, and should return the total amounts of profits/losses/debt payments (in `want` tokens) for the Vault's accounting. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- _adjustPosition: internal method that performs any adjustments to the core position(s) of this Strategy given what change the Vault made in the "investable capital" available to the Strategy. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- _liquidatePosition: internal method that liquidates up to `amountNeeded_` of `want` of this strategy's positions, regardless of slippage. Any excess will be re-invested with `_adjustPosition()`. This function should return the amount of `want` tokens made available by the liquidation. If there is a difference between them, `loss_` indicates whether the difference is due to a realized loss or if there is some other situation at play (e.g. locked funds) where the amount made available is less than what is needed. This function should be

overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.

- _liquidateAllPositions: internal method that liquidates everything and returns the amount that got freed. This function is used during emergency exit instead of `_prepareReturn()` to liquidate all of the Strategy's positions back to the Vault. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- _prepareMigration: internal method that does anything necessary to prepare this Strategy for migration, such as transferring any reserve or LP tokens, CDPs, or other tokens or stores of value. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- _protectedTokens: internal method that overrides this to add all tokens/tokenized positions this contract manages on a persistent basis. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.
- _governance: internal method that returns the address of vault governance.
- _onlyAuthorized: internal method that checks if msg.sender has this permission. This permission includes these roles: strategist, vault's governance.
- _onlyEmergencyAuthorized: internal method that checks if msg.sender has this permission. This permission includes these roles: strategist, vault's governance, vault's guardian, vault's management.
- _onlyStrategist: internal method that checks if msg.sender has this permission. This permission includes these roles: strategist.
- _onlyGovernance: internal method that checks if msg.sender has this permission. This permission includes these roles: vault's governance.
- _onlyKeepers: internal method that checks if msg.sender has this permission. This permission includes these roles: keeper, strategist, vault's governance, vault's guardian, vault's management.

- _onlyVaultManagers:internal method that checks if msg.sender has this permission. This permission includes these roles: vault's management, vault's governance.

2.2.4. GenericLenderBase.sol

An abstract upgradeable contract that implements the base functionality required for a lending platform. This contract should be inherited, and the abstract methods should be implemented to adapt the Lender to the particular needs it has to create a return.

2.2.4.1. Assets

The contract contains the following entities:

- vault: The address of the upstream “Vault” contract to be connected.
- strategy: The address of the “Strategy” contract to be connected.
- want: ERC20 token address of the invest token.
- lenderName: String that lender's name.
- dust: The value that used to ignore particularly small amounts of assets in the contract.

2.2.4.2. Modifiers

The contract has the following modifiers:

- **`onlyGovernance:`** Checks if msg.sender has this permission. This permission includes these roles: vault's governance.
- **`management:`** Checks if msg.sender has this permission. This permission includes these roles: vault's governance, vault's management, the address of Strategy contract.

2.2.4.3. Functions

The contract has the following functions:

- __GenericLenderBase_init: internal method that initializes the Lender. This is called only once when the contract is deployed.Call method __GenericLenderBase_init_unchained().
- __GenericLenderBase_init_unchained: internal method that initializes the Lender's personalized configuration. This is called only once when the contract is deployed. Sets these variables: strategy, vault, want, lenderName, dust.
- setDust: sets up the new dust value. Only "management" can call this function.
- sweep: removes tokens from this Lender that are not the type of tokens managed by this Lender. This may be used in case of accidentally sending the wrong kind of token to this Lender. Only "management" can call this function.
- _protectedTokens: internal method that overrides this to add all tokens/tokenized positions this contract manages on a persistent basis. This function should be overridden, and the implementation logic should be modified to adapt the strategy to its specific needs.

2.2.5. WETHStrategyToLido.sol

A WETH investment strategy contract that earns income by investing WETH into Lido. The contract inherits from BaseStrategy, and it is an upgradeable contract.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract, their explanation can be found in the Functions header:

- **name()**
- **ethToWant()**
- **estimatedTotalAssets()**
- **estimatedPotentialTotalAssets()**

- **wantBalance()**

- **stethBalance()**

2.2.5.1. Assets

The contract contains the following entities:

- STABLE_SWAP_STETH: Constant address of CurveFi.
- WETH: Constant address of WETH.
- STETH: Constant address of stETH.
- DENOMINATOR: Constant denominator for %100.
- _WETHID: Constant id of WETH in CurveFi.
- _STETHID: Constant id of stETH in CurveFi.
- _referal: Referrer address when staking to Lido.
- maxSingleTrade: Maximum single transaction amount.
- slippageProtectionOut: Slippage value when swapping in CurveFi.
- reportLoss: Mark whether to report when a loss occurs.
- dontInvest: Mark whether investment behavior is prohibited.
- peg: Percentage value, a certain percentage of stETH is reserved to deal with the risks caused by exchange rate fluctuations.

2.2.5.2. Functions

The contract has the following functions:

- initialize: initializes the Strategy. Call methods __BaseStrategy_init() and __WETHStrategyToLido_init_unchained().
- __WETHStrategyToLido_init_unchained: internal method that initializes the Strategy. This is called only once when the contract is deployed. Sets these variables: maxReportDelay, profitFactor, debtThreshold, _referal, maxSingleTrade, slippageProtectionOut, peg.
- updateReferal: sets up the new _referal value. "onlyEmergencyAuthorized" can call this function.
- updateMaxSingleTrade: sets up the new maxSingleTrade value. "onlyVaultManagers" can call this function.

- updatePeg: sets up the new peg value. "onlyVaultManagers" can call this function.
- updateReportLoss: sets up whether to report when a loss occurs. "onlyVaultManagers" can call this function.
- updateDontInvest: sets up whether investment behavior is prohibited. "onlyVaultManagers" can call this function.
- updateSlippageProtectionOut: sets up the new slippageProtectionOut value. "onlyVaultManagers" can call this function.
- invest: Invest the specified amount of WETH in Strategy into Lido. "onlyEmergencyAuthorized" can call this function.
- rescueStuckEth: Convert ETH in Strategy to WETH. "onlyEmergencyAuthorized" can call this function.
- name: returns Strategy's name.
- estimatedTotalAssets: returns an accurate estimate for the total amount of assets that this Strategy is currently managing. The estimated total assets are the total balance of stETH and WETH minus the amount of stETH used to cover the risk of exchange rate fluctuations.
- estimatedPotentialTotalAssets: returns the total current balance of stETH and WETH in Strategy.
- wantBalance: returns the current balance of stETH in Strategy.
- stethBalance: returns the current balance of stETH in Strategy.
- ethToWant: returns the passed in value. The exchange ratio of WETH and ETH is 1:1.
- _prepareReturn: internal method that overrides the BaseStrategy method. Implemented the logic of income settlement and reinvestment for Lido investment strategy.
- _liquidateAllPositions: internal method that overrides the BaseStrategy method. Exchange all stETH in the strategy into WETH through CurveFi.
- _adjustPosition: internal method that overrides the BaseStrategy method. Call _invest() to invest WETH into Lido.

- _invest: internal method that overrides the BaseStrategy method. Choose between staking to Lido or exchanging for stETH in CurveFi to maximize your benefits.
- _divest: internal method that overrides the BaseStrategy method. Withdraw by exchanging WETH for stETH in CurveFi.
- _liquidatePosition: internal method that overrides the BaseStrategy method. Exchange the specified amount of the strategy into WETH through CurveFi.
- _prepareMigration: internal method that overrides the BaseStrategy method. Transfer all stETH to the new strategy.
- _protectedTokens: internal method that overrides the BaseStrategy method. stETH is protected.
- _stableSwapSTETH: internal method that returns the address of CurveFi contract.
- _weth: internal method that returns the address of WETH token.
- _stETH: internal method that returns the address of stETH token.

2.2.6. OJEEStrategyToFarm.sol

A OJEE investment strategy contract that earns income by investing OJEE into ERC20TokenFarmPool. The contract inherits from BaseStrategy, and it is an upgradeable contract.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract, their explanation can be found in the Functions header:

- **name()**
- **ethToWant()**
- **estimatedTotalAssets()**
- **estimatedAPR()**
- **strategyBalanceAndRewardInPool()**

- **wantBalance()**

- **stethBalance()**

2.2.6.1. Assets

The contract contains the following entities:

- SECONDSPERYEAR: Constant number of seconds in a year.
- tokenFarmPool: The address of ERC20TokenFarmPool contract.
- withdrawalThreshold: Minimum amount to withdraw assets from the lenders.

2.2.6.2. Functions

The contract has the following functions:

- initialize: initializes the Strategy. Call methods __BaseStrategy_init() and __OJEEStrategyToFarm_init_unchained().
- __OJEEStrategyToFarm_init_unchained: internal method that initializes the Strategy. This is called only once when the contract is deployed. Sets these variables: maxReportDelay, profitFactor, debtThreshold, tokenFarmPool.
- setWithdrawalThreshold: sets up the new withdrawalThreshold value. "onlyAuthorized" can call this function.
- setFarmPool: sets up the address of ERC20TokenFarmPool contract. "onlyAuthorized" can call this function.
- name: returns Strategy's name.
- harvestTrigger: The method overrides the BaseStrategy method. We have completed the amount conversion of ETH and OJEE before sending the transaction. The param `callCostInWant_` is different from BaseStrategy's `callCostInWei`. `callCostInWant_` must be priced in terms of `want`. And `callCostInWei` must be priced in terms of `wei` (1e-18 ETH).

- ethToWant: returns the passed in value. Because we have completed the quantity conversion of ETH and OJEE before sending the transaction, we only need to return it at 1:1 here.
- estimatedTotalAssets: Get an estimate of the total assets held by the Strategy. totalAsset=Strategy's principal and income in ERC20TokenFarmPool + Strategy's OJEE balance.
- estimatedAPR: Get an estimate of the annual percentage rate (APR) for the Strategy.
- strategyBalanceAndRewardInPool: Get the balance and earned rewards held by the Strategy in the pool.
- _prepareReturn: internal method that overrides the BaseStrategy method. Implemented the logic of income settlement and reinvestment for ERC20TokenFarmPool investment strategy.
- _adjustPosition: internal method that overrides the BaseStrategy method. Deposit all OJEE token in Strategy to ERC20TokenFarmPool.
- _withdrawSome: Withdraw the specified amount of assets from ERC20TokenFarmPool.
- _withdrawAll: Withdraw all investment assets and earnings from ERC20TokenFarmPool.
- _liquidatePosition: internal method that overrides the BaseStrategy method. Withdraw the specified amount of assets from ERC20TokenFarmPool.
- _liquidateAllPositions: internal method that overrides the BaseStrategy method. Withdraw all investment assets and earnings from ERC20TokenFarmPool.
- _prepareMigration: internal method that overrides the BaseStrategy method. Withdraw all investment assets from ERC20TokenFarmPool.
- _protectedTokens: internal method that overrides the BaseStrategy method. No additional tokens are protected.

2.2.7. GenericAaveV3.sol

A USDT investment lender contract that earns income by investing USDT into AaveV3. The contract inherits from GenericLenderBase, and it is an upgradeable contract.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract, their explanation can be found in the Functions header:

- **nav()**
- **apr()**
- **weightedApr()**
- **aprAfterDeposit()**
- **hasAssets()**
- **underlyingBalanceStored()**

2.2.7.1. Assets

The contract contains the following entities:

- _DEFAULT_REFERRAL: Default referral code for investing in Aave.
- PROTOCOL_DATA_PROVIDER: Constant address of Aave's ProtocalDataProvider contract.
- aToken: The address of the certificate token obtained by pledging USDT to AaveV3.
- _customReferral: Invest in Aave’s custom referral code.

2.2.7.2. Functions

The contract has the following functions:

- initialize: initializes the contract. Call methods __GenericLenderBase_init() and __GenericAaveV3_init_unchained().

- __GenericAaveV3_init_unchained: internal method that initializes the contract. This is called only once when the contract is deployed. Sets the address of aToken.
- setReferralCode: sets up the new _customReferral value. Only "management" can call this function.
- deposit: deposit the USDT in the current render contract into AaveV3. Call methods _deposit(). Only "management" can call this function.
- withdraw: withdraw a specified amount of assets from AaveV3. Call methods _withdraw(). Only "management" can call this function.
- emergencyWithdraw: in an emergency, withdraw a specified amount of assets from AaveV3. "onlyGovernance" can call this function.
- withdrawAll: withdraw all assets from AaveV3. Only "management" can call this function.
- nav: returns the total assets currently managed by the lender. Call methods _nav().
- apr: returns the current investment apr. Call methods _apr().
- weightedApr: returns the amount of revenue that can be generated under the current apr.
- aprAfterDeposit: returns the new apr after deposit `extraAmount_` 'want' token.
- hasAssets: returns whether there are any investment assets in the contract
- underlyingBalanceStored: returns the total assets and earnings invested in AaveV3.
- _withdraw: withdraw a specified amount of assets from AaveV3.
- _deposit: deposit the USDT in the current render contract into AaveV3.
- _nav: internal method that returns the total assets currently managed by the lender.
- _apr: internal method that returns the current investment apr.
- _lendingPool: internal method that returns the address of AaveV3's USDT lending pool contract.
- _protectedTokens: internal method that overrides the GenericLenderBase method. `want` and aToken are protected.

- _protocolDataProvider: internal method that returns the address of AaveV3's ProtocalDataProvider contract.

2.2.8. USDTStrategyToLender.sol

A USDT investment strategy contract that earns income by investing USD into multiple lending platforms. The contract inherits from BaseStrategy, and it is an upgradeable contract.

A typical flow looks as follows:

Contract deployment

When deploying the USDTStrategyToLender contract, associate the vault contract.

Add lenders for USDTStrategyToLender

2. Deploy the GenericAaveV3 contract bound to AaveV3 and associate it with the USDTStrategyToLender contract.
3. Call addLender() of USDTStrategyToLender to add the deployed GenericAaveV3 contract to the lenders array of USDTStrategyToLender.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract, their explanation can be found in the Functions header:

- **name()**
- **ethToWant()**
- **estimatedTotalAssets()**
- **tendTrigger()**
- **lendStatuses()**
- **numLenders()**

- **estimatedAPR()**

- **estimateAdjustPosition()**

- **estimatedFutureAPR()**

- **lentTotalAssets()**

2.2.8.1. Assets

The contract contains two Structs:

- **LenderRatio:** This structure holds the asset proportion information of GenericLender.
  - address lender - address of added GenericLender contract.
  - uint16 share - The ratio of the assets managed by this lender to the investment assets. 100% is 1000.
- **LendStatus:** This structure holds the investing status of the GenericLender contract.
  - string name - the GenericLender's name.
  - uint256 assets - the total assets currently managed by this lender.
  - uint256 rate - the apr of theGenericLender contract.
  - address add - address of the GenericLender contract..

Besides the mentioned structs, the following entities are present in the project:

- UNISWAP_ROUTER: Constant address of UniswapV2.
- WETH: Constant address of WETH.:
- SECONDSPERYEAR: Constant number of seconds in a year.
- withdrawalThreshold: Minimum amount to withdraw assets from the lenders.
- lenders: A public GenericLender array holding the address of the added GenericLender.
- wantToEthOracle: The address of WantToEth contract.

2.2.8.2. Functions

The contract has the following functions:

- initialize: initializes the Strategy. Call methods __BaseStrategy_init() and __USDTStrategyToLender_init_unchained().
- __USDTStrategyToLender_init_unchained: internal method that initializes the Strategy. This is called only once when the contract is deployed. Sets these variables: maxReportDelay, profitFactor, debtThreshold.
- setWithdrawalThreshold: sets up the new withdrawalThreshold value.
- setPriceOracle: sets up the address of WantToEthOracle contract.
- name: returns Strategy's name.
- addLender: add lenders for the strategy to choose from.
- safeRemoveLender: remove lenders safely. If an abnormality occurs in the asset extraction of AaveV3, resulting in the assets extracted by the lender from AaveV3 being lower than the expected withdrawal amount, the transaction will revert and the lenders can not be removed.
- forceRemoveLender: forcibly remove lenders. If an abnormality occurs in the asset extraction of AaveV3, causing the asset extracted by the lender from AaveV3 to be lower than the expected withdrawal amount, the transaction will continue and the lenders can still be removed.
- manualAllocation: redistribute the proportion of assets that can be managed by each lender in the lenders array.
- ethToWant: returns an accurate conversion from `amtInWei_` ETH to `want`. If `want` is WETH, return no change. If wantToEthOracle is not 0x00, use the external oracle. Else use Uniswap swap price.
- tendTrigger: returns whether tend() should be called currently. Let's check if there is a better APR somewhere else. If there is a profit potential worth changing, then to calculate our potential profit increase, we work out how much extra we would make in a typical harvest interlude. That is maxReportingDelay then we see if the extra profit is worth more than the gas cost * profitFactor.
- lendStatuses: returns the status of all lenders attached to the strategy.

- estimatedTotalAssets: Get an estimate of the total assets held by the Strategy. totalAsset=total assets managed by all lenders + Strategy's USDT balance.
- numLenders: returns the number of lenders in array type.
- estimatedAPR: returns the average apr of all lenders.
- estimateAdjustPosition: Estimates the lenders with the highest and lowest APR. Returns the apr and index of the lenders with the lowest and highest APR.
- estimatedFutureAPR: returns an estimate of future APR with a change in the debt limit.
- lentTotalAssets: returns the total number of assets that can be withdrawn across all lenders.
- _prepareReturn: internal method that overrides the BaseStrategy method. Implemented the logic of income settlement and reinvestment for AaveV3 investment strategy.
- _adjustPosition: internal method that overrides the BaseStrategy method. The algorithm moves assets from lowest return to highest, like a very slow idiots bubble sort. We ignore debt outstanding for an easy life.
- _withdrawSome: withdraw the specified amount of assets from all lenders. Prioritize assets from the lender with the lowest APR. Cycle through withdrawals starting with the worst rate.
- _liquidatePosition: internal method that overrides the BaseStrategy method. Withdraw the specified amount of assets from all lenders.
- _removeLender: remove lender from lenders array.
- _liquidateAllPositions: internal method that overrides the BaseStrategy method. Liquidate all positions by withdrawing from the worst rate first.
- _prepareMigration: internal method that overrides the BaseStrategy method. Withdraw all investment assets from lenders.
- _uniswapRouter: internal method that returns the address of UniswapV2 contract.
- _weth: internal method that returns the address of WETH token.

- _estimateDebtLimitIncrease: internal method that returns the estimated impact on APR if we add more funds.
- _estimateDebtLimitDecrease: internal method that returns the estimated impact on APR if the debt limit decreases.
- _protectedTokens: internal method that overrides the BaseStrategy method. No additional tokens are protected.