DPEX

SMART CONTRACT AUDIT

zokyo

January 23rd 2023 | v. 1.0

Security Audit Score

PASS

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

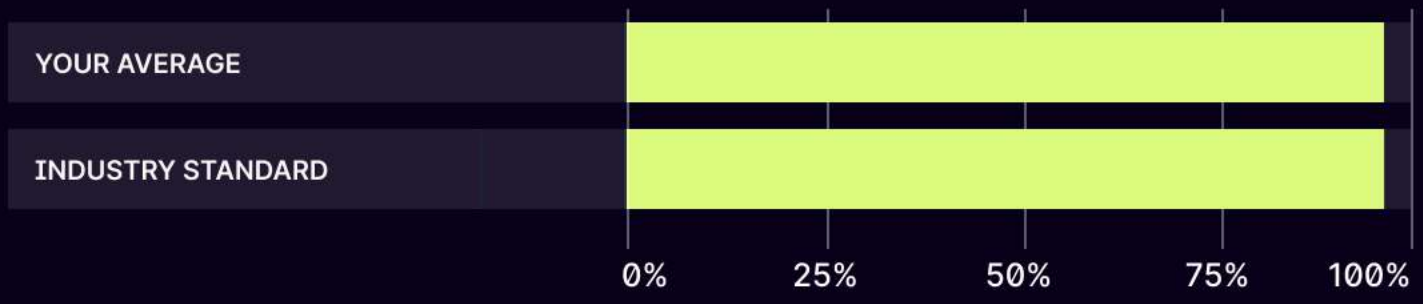
TECHNICAL SUMMARY

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

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

Contract Status

Testable Code



All the contracts are testable, a sufficient test-coverage was provided both by the DPEX team and Zokyo Security team.
It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the DPEX team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the DPEX repository:
https://github.com/DPEX-io/dpex

Intial commit: develop branch, f0df642cfa9b930a79d561ad1b68f9bc352ecbf1
Final commit: develop branch, 320887ccf885cc1a9b1a058df113c65790715125

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

- DPEX.sol
- DPLP.sol
- BaseToken.sol
- MintableBaseToken.sol
- OrderBookReader.sol
- OrderBook.sol
- PositionManager.sol
- BasePositionManager.sol
- Governable.sol
- RewardTracker.sol
- RewardRouter.sol
- Vault.sol
- DPLPManager.sol

During the audit, Zokyo Security ensured that the contract:

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

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

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

Executive Summary

Zokyo Security team has conducted an audit of the DPEX protocol. DPEX represents a decentralized perpetual exchange for trading of assets and staking for earning additional rewards. All the smart-contracts within the scope were carefully checked during the audit. The goal of the audit was to analyze the security of smart contracts, ensure that they correspond to Solidity best practices in terms of code style and gas expenditure, validate the security of users' funds and correspond to the business logic of the protocol. While the auditors have carefully checked all the contracts within the scope, there were several crucial contracts that were out of the current scope. Thus, the auditors can't be completely sure in the correctness of such contracts as VaultPriceFeed.sol and VaultUtils.sol.

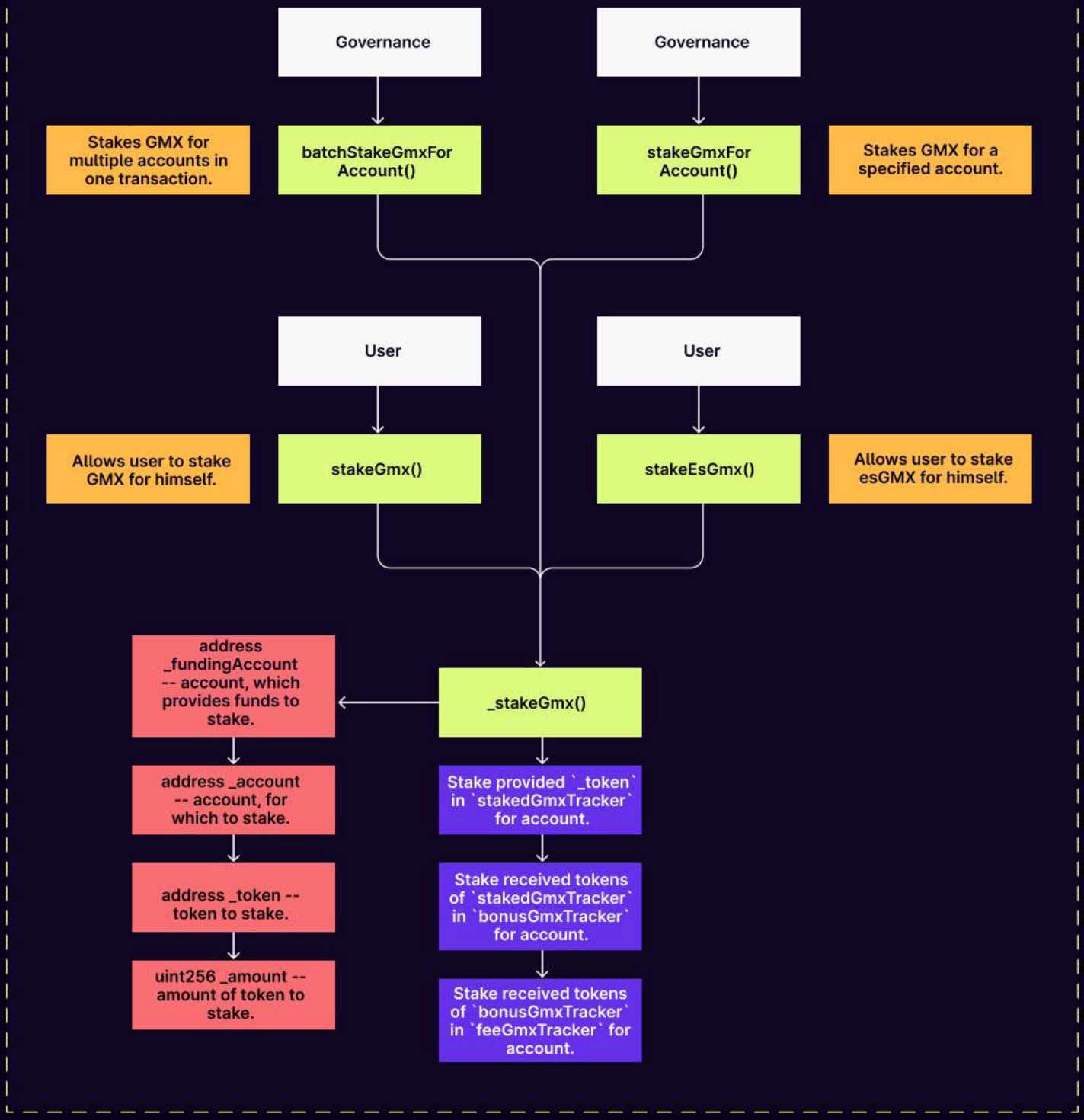
There were several high- and medium-severity issues found during the manual audit. One of the high-severity issues was connected to the absence of .send() validation during ETH transferring, which could make the funds stuck. The issue was fixed by the DPEX team: .call() is used now, and the result of the transfer is validated. Other high-severity issues were connected to the access of privileged accounts to users' funds. Both of them were also fixed by the DPEX team. Medium-severity issues were connected to the possible blocking of withdrawing users' funds, safety during the vault upgrades, ERC20 transfer validation, outdated Solidity version, and the correctness of a variable validation in the setter. All of these issues were successfully fixed or verified by the DPEX team. One medium-severity issue is worth mentioning, though. This is Medium-2, which is connected to the blocking of the underlying funds for any RewardTracker tokens that were transferred. The issue describes the following functionality: in case RewardTracker tokens are transferred from one account to another, they couldn't be unstaked until the tokens are transferred back. The DPEX team has acknowledged the issue and assured that users of the dApp will be notified to be cautious when transferring RewardTracker tokens. However, due to the future updates of the contract, the issue could not be fixed at the moment. Other issues were connected to the absence of variable validations, events in setters, the upgradability of the contracts, and other logic validations. All of them were fixed or verified by the DPEX team as well. A detailed overview of all issues can be seen in the complete analysis section.

The contracts are well-written, though they lack natspec documentation. The overall security of the protocol is high enough, but some of the contracts containing the essential logic were out of the audit scope.

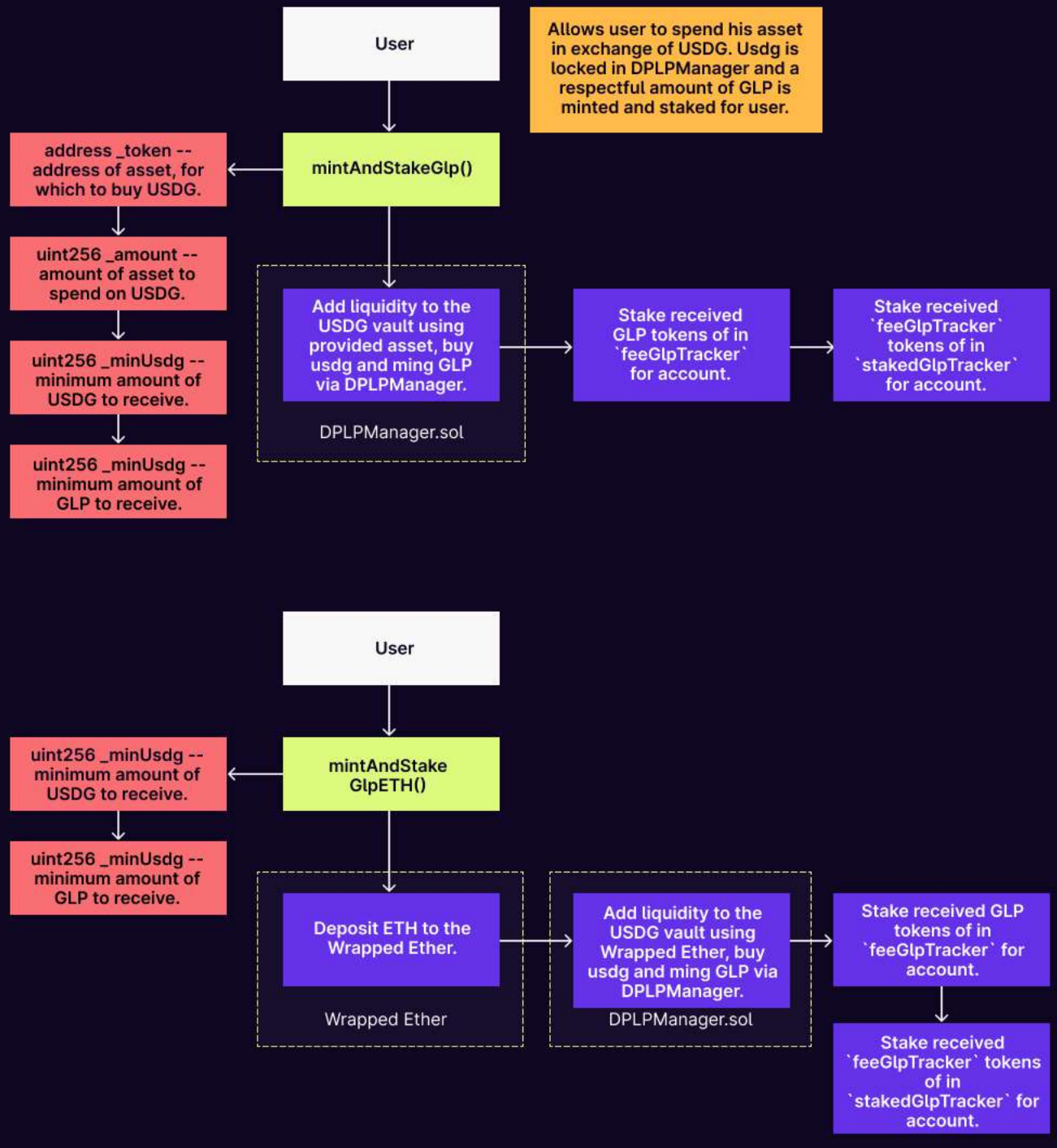
The DPEX team has also prepared a sufficient set of tests that validate the business logic of the contracts. The team has also prepared documentation on the protocol https://docs.dpex.io/. Nevertheless, Zokyo Security has prepared their own set of unit tests to validate the correctness of the contracts and safety of users' funds.

Also some of the minor changes were reverted due to size limit of smart-contracts. Thus, Medium-3, part of Low-1 and Info-2 were reverted for Vault.sol. Despite a low threat level of the issues, reverting them still decreases the secuirty level of the smart-contract, which is why auditors still recommend to search for other ways of optimizing the size of smart-contract such as splitting the smart-contracts or putting part of the logic in the library. Also, as for the Medium-3 issue, the DPEX team has assured, that during the deployment, a Timelock smart-contract will be used behind the Governance to ensure the safety of all the actions of the Governance. The address of Timelock will be provided once protocol is deployed.

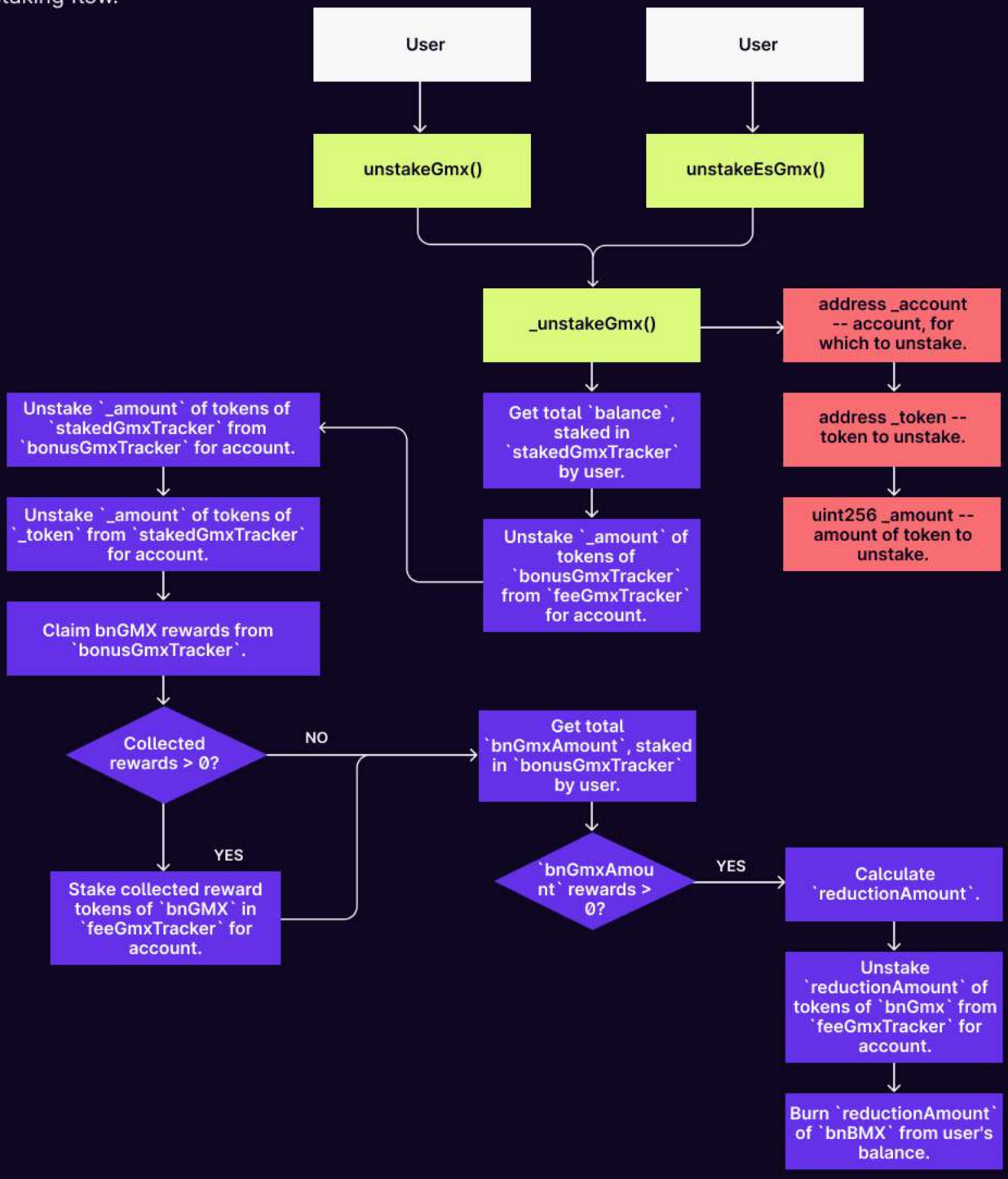
Staking flow.



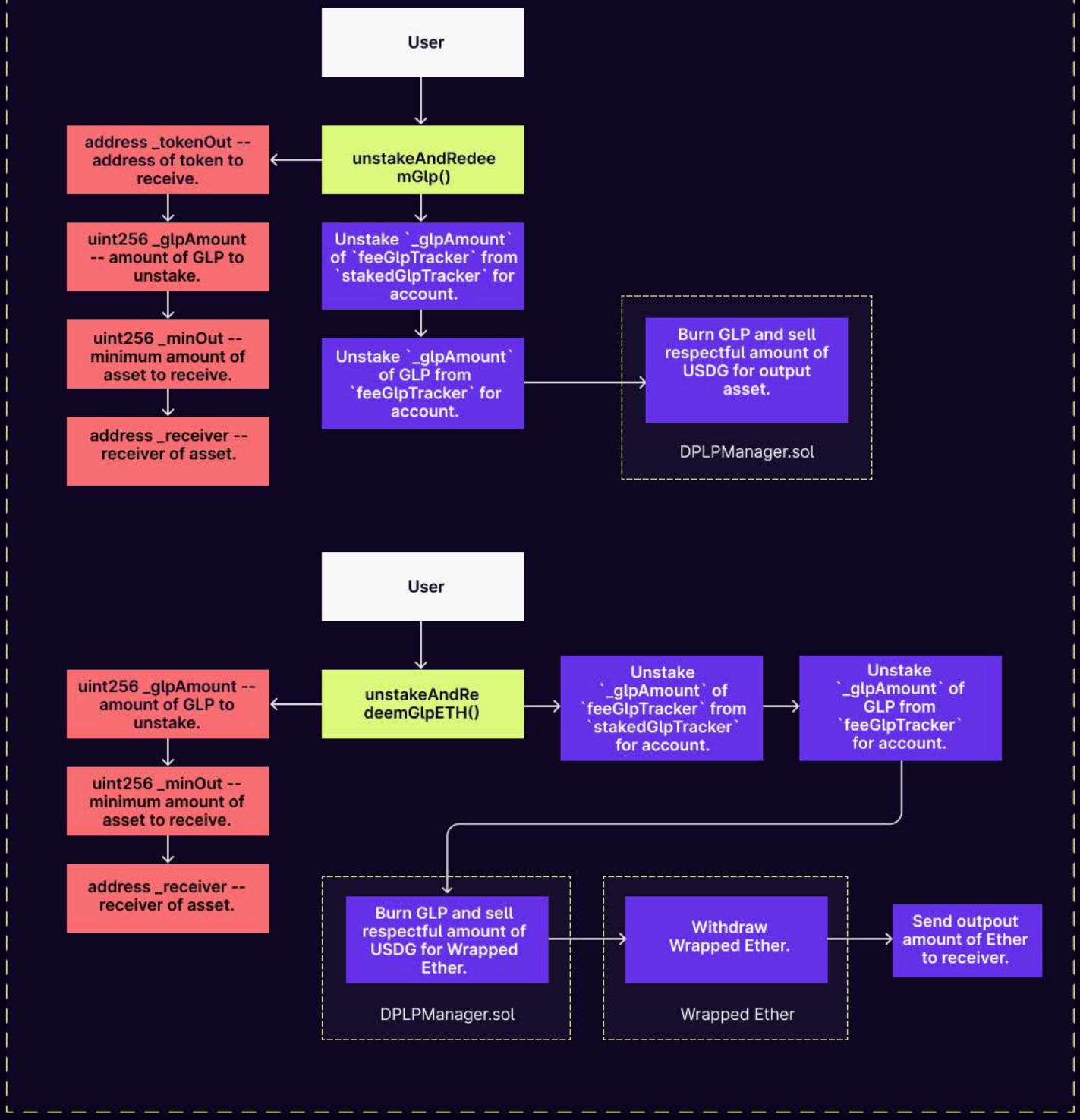
Staking flow.



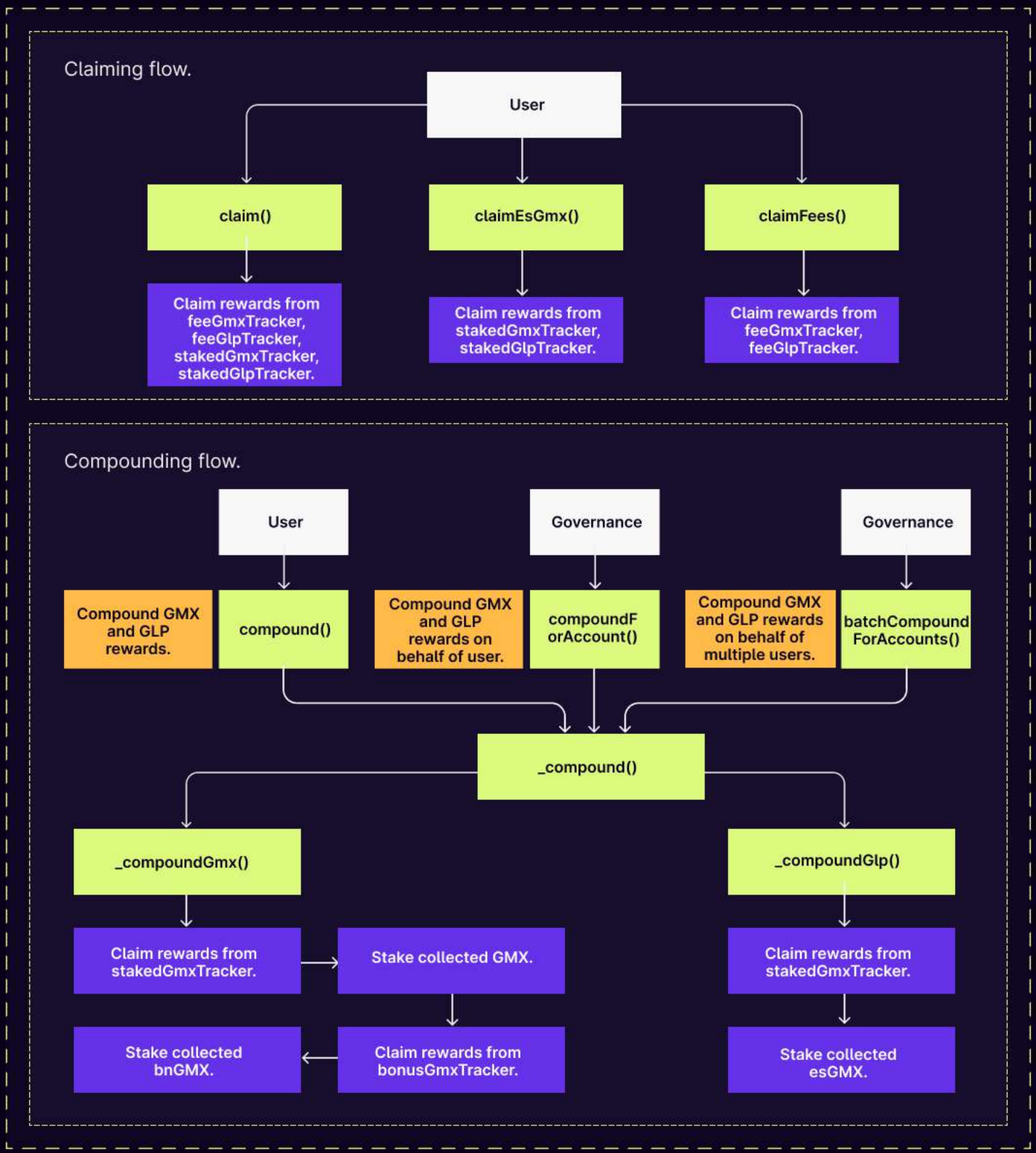
Unstaking flow.



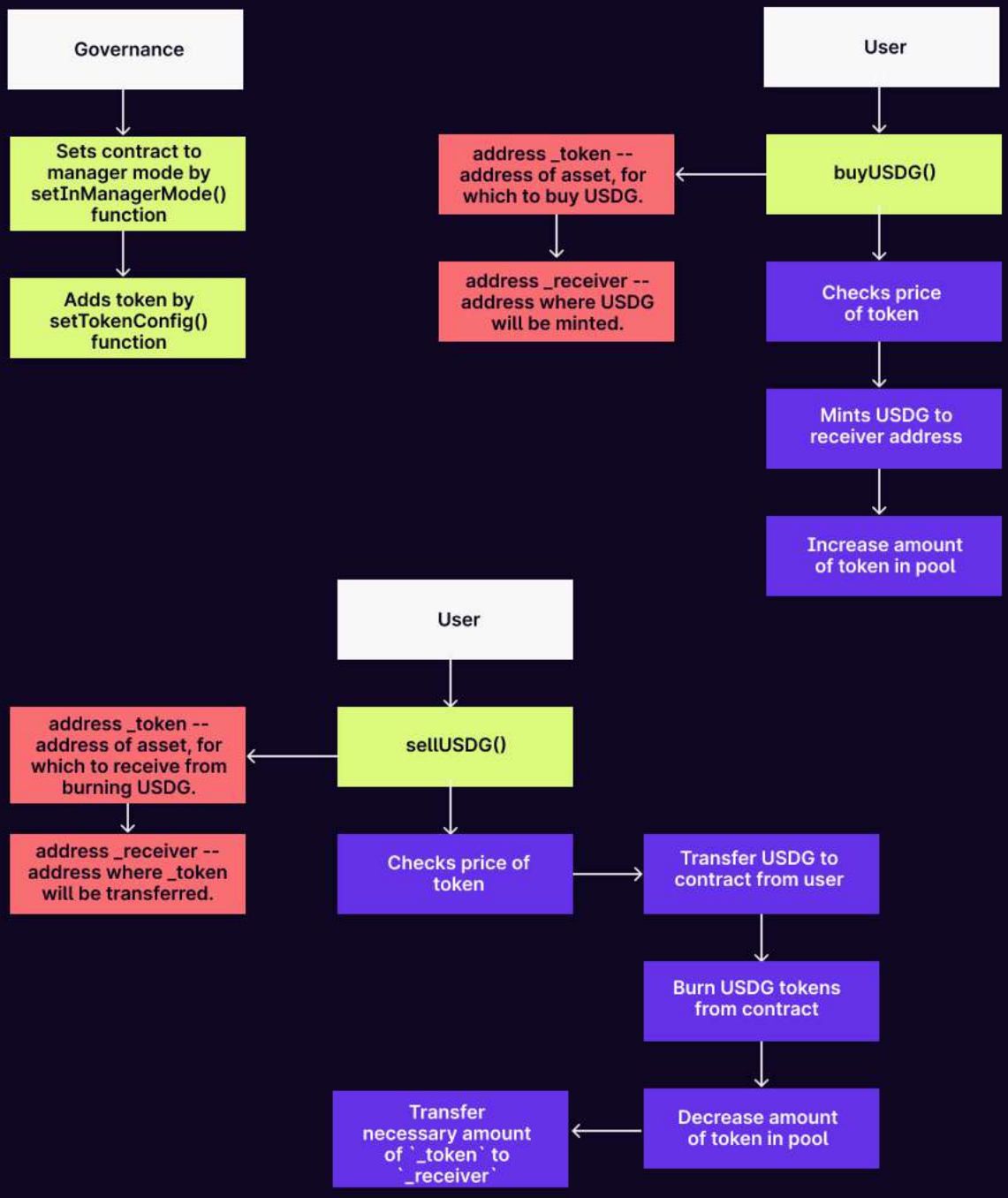
Unstaking flow.



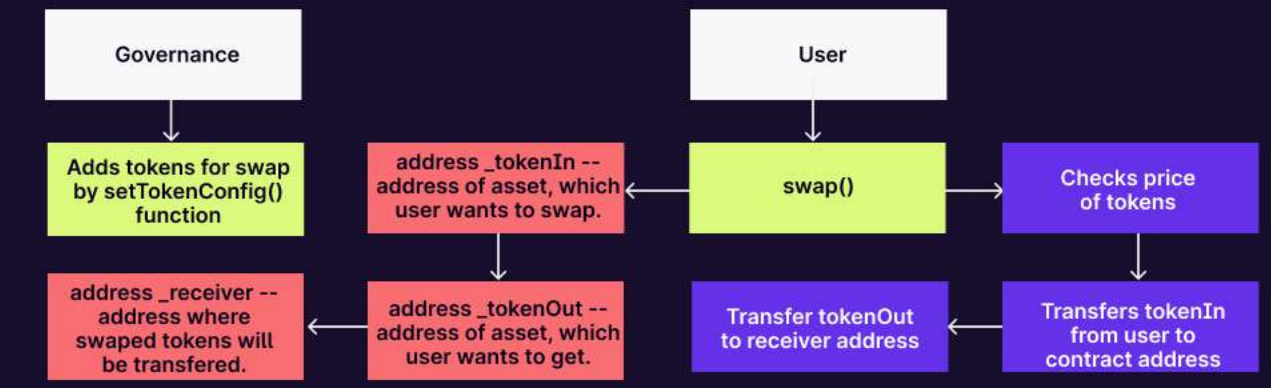
REWARDROUTER.SOL



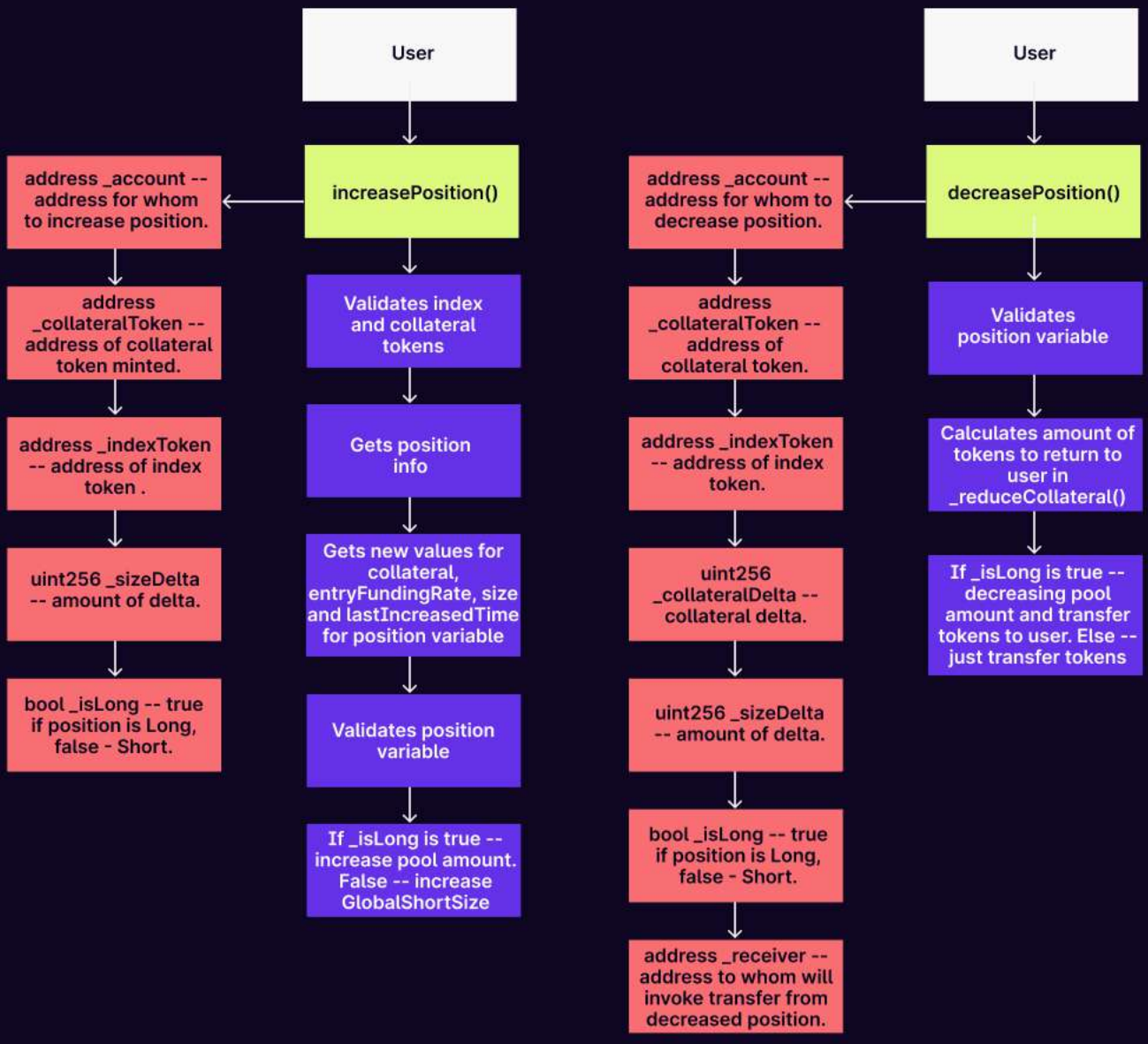
Buy/sell Usdg flow.

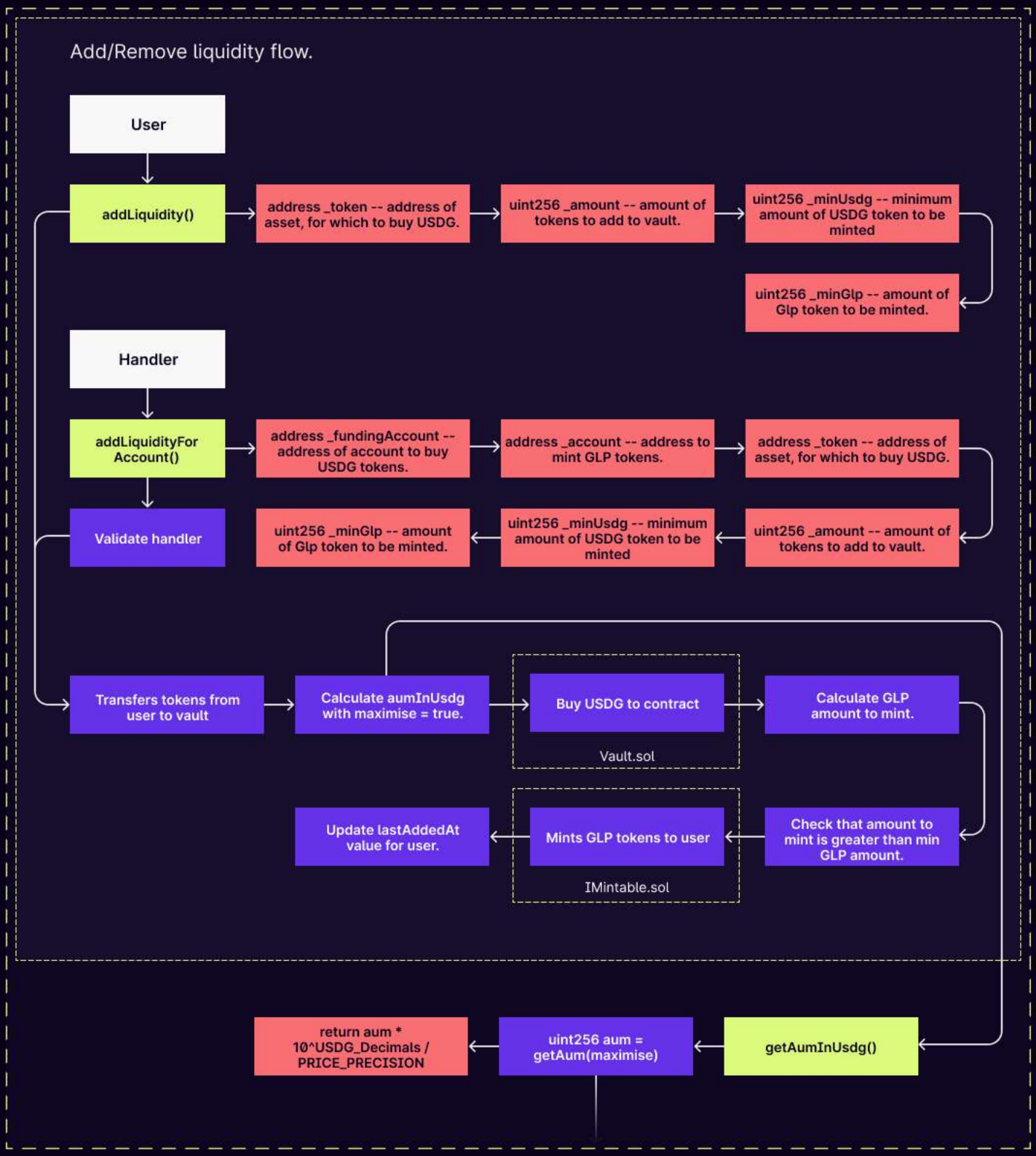


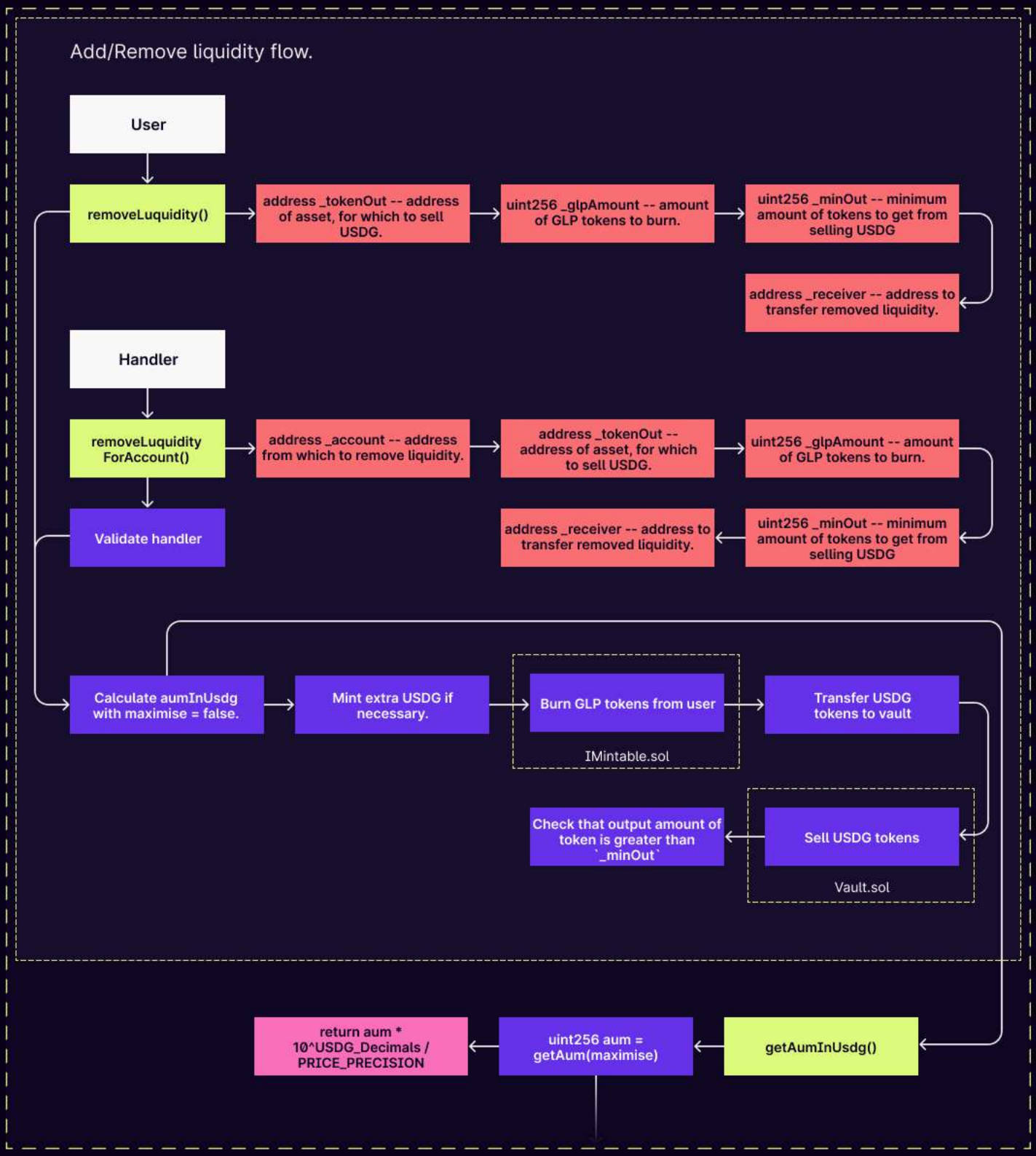
Swap flow.


Governance
User
Adds tokens for swap by setTokenConfig() function
address _tokenIn -- address of asset, which user wants to swap.
swap()
Checks price of tokens
address _receiver -- address where swaped tokens will be transfered.
address _tokenOut -- address of asset, which user wants to get.
Transfer tokenOut to receiver address
Transfers tokenIn from user to contract address

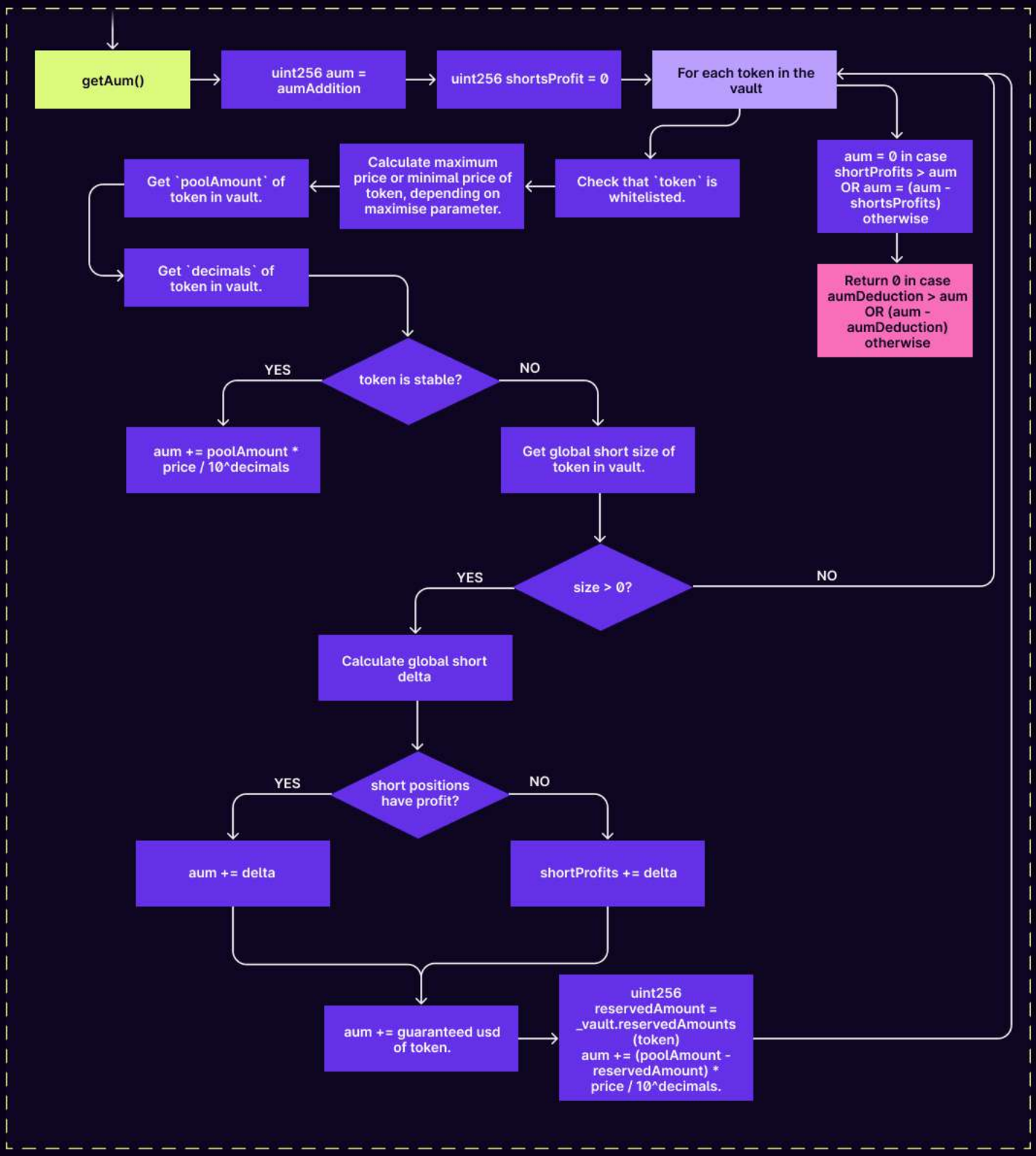
Increase/Decrease position flow.




Add/Remove liquidity flow.
User
addLiquidity()
address _token -- address of asset, for which to buy USDG.
uint256 _amount -- amount of tokens to add to vault.
uint256 _minUsdg -- minimum amount of USDG token to be minted
uint256 _minGlp -- amount of Glp token to be minted.
Handler
addLiquidityFor Account()
address _fundingAccount -- address of account to buy USDG tokens.
address _account -- address to mint GLP tokens.
address _token -- address of asset, for which to buy USDG.
Validate handler
uint256 _minGlp -- amount of Glp token to be minted.
uint256 _minUsdg -- minimum amount of USDG token to be minted
uint256 _amount -- amount of tokens to add to vault.
Transfers tokens from user to vault
Calculate aumInUsdg with maximise = true.
Buy USDG to contract
Vault.sol
Calculate GLP amount to mint.
Update lastAddedAt value for user.
Mints GLP tokens to user
IMintable.sol
Check that amount to mint is greater than min GLP amount.
return aum * 10^USDG_Decimals / PRICE_PRECISION
uint256 aum = getAum(maximise)
getAumInUsdg()

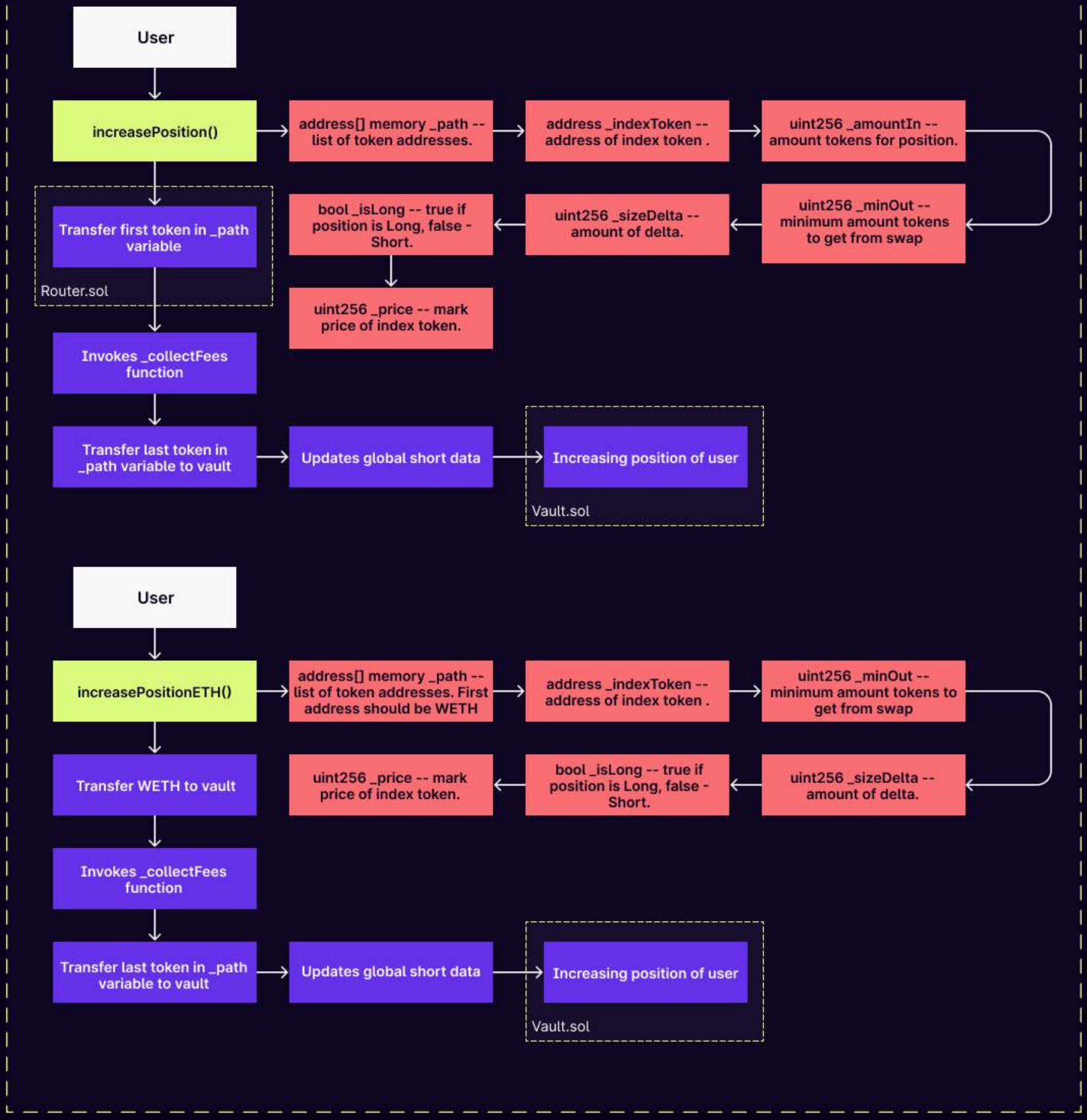

Add/Remove liquidity flow.
User
removeLuquidity()
address _tokenOut -- address of asset, for which to sell USDG.
uint256 _glpAmount -- amount of GLP tokens to burn.
uint256 _minOut -- minimum amount of tokens to get from selling USDG
address _receiver -- address to transfer removed liquidity.
Handler
removeLuquidity ForAccount()
address _account -- address from which to remove liquidity.
address _tokenOut -- address of asset, for which to sell USDG.
uint256 _glpAmount -- amount of GLP tokens to burn.
Validate handler
address _receiver -- address to transfer removed liquidity.
uint256 _minOut -- minimum amount of tokens to get from selling USDG
Calculate aumInUsdg with maximise = false.
Mint extra USDG if necessary.
Burn GLP tokens from user
IMintable.sol
Transfer USDG tokens to vault
Check that output amount of token is greater than `_minOut`
Sell USDG tokens
Vault.sol
return aum * 10^USDG_Decimals / PRICE_PRECISION
uint256 aum = getAum(maximise)
getAumInUsdg()

DPLPMANAGER.SOL



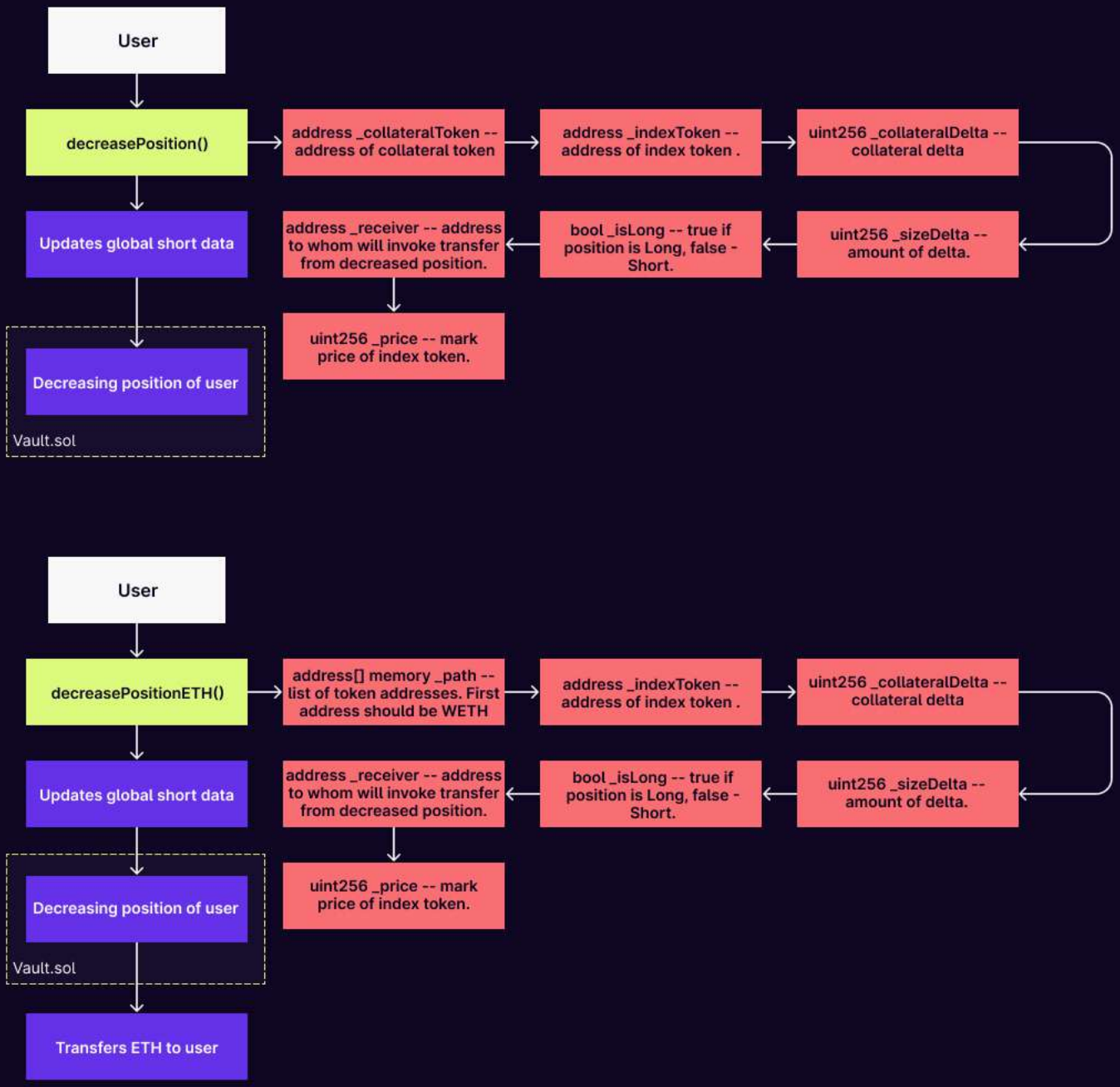
POSITIONMANAGER.SOL

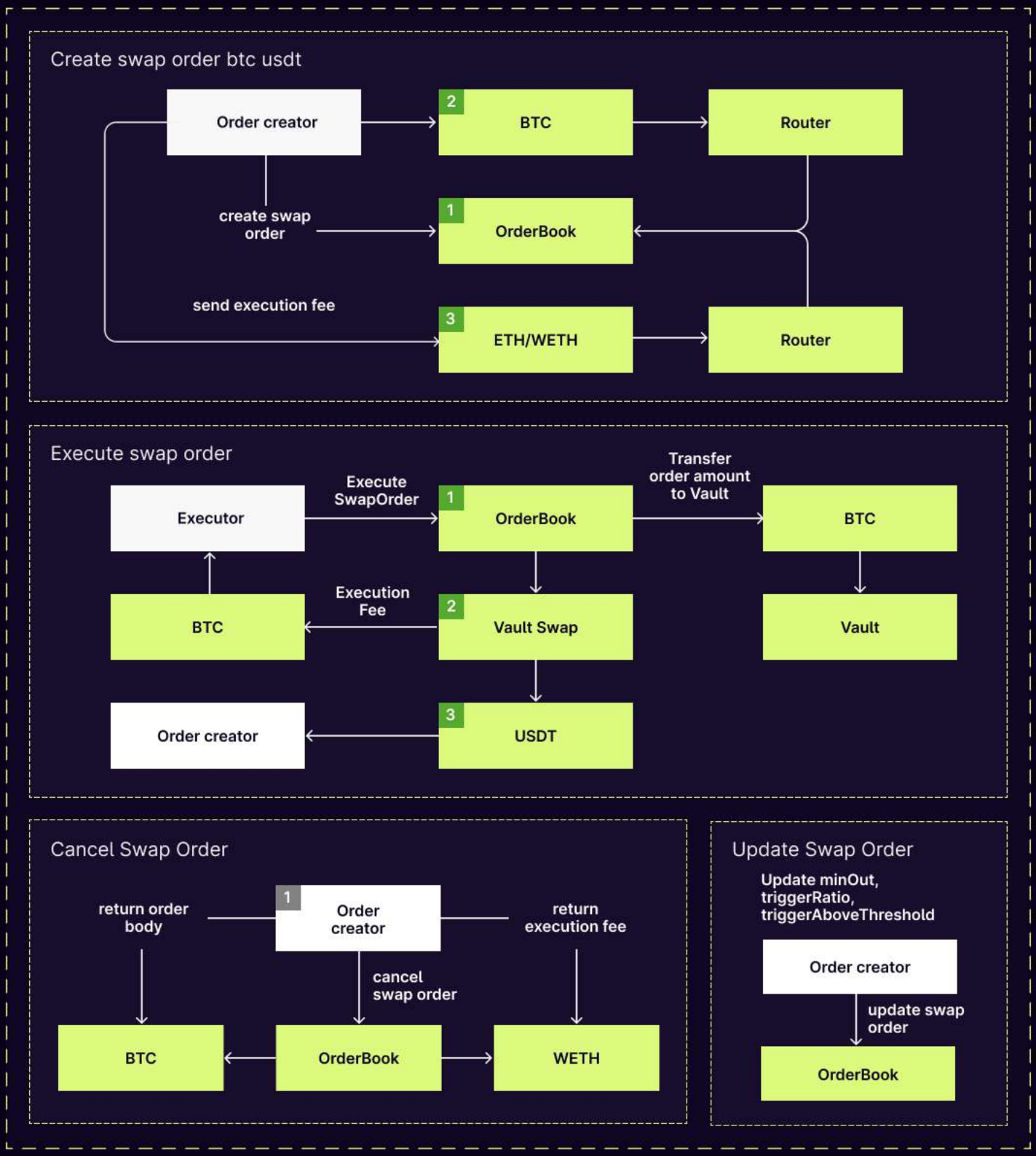
Increase/decrease position flow.

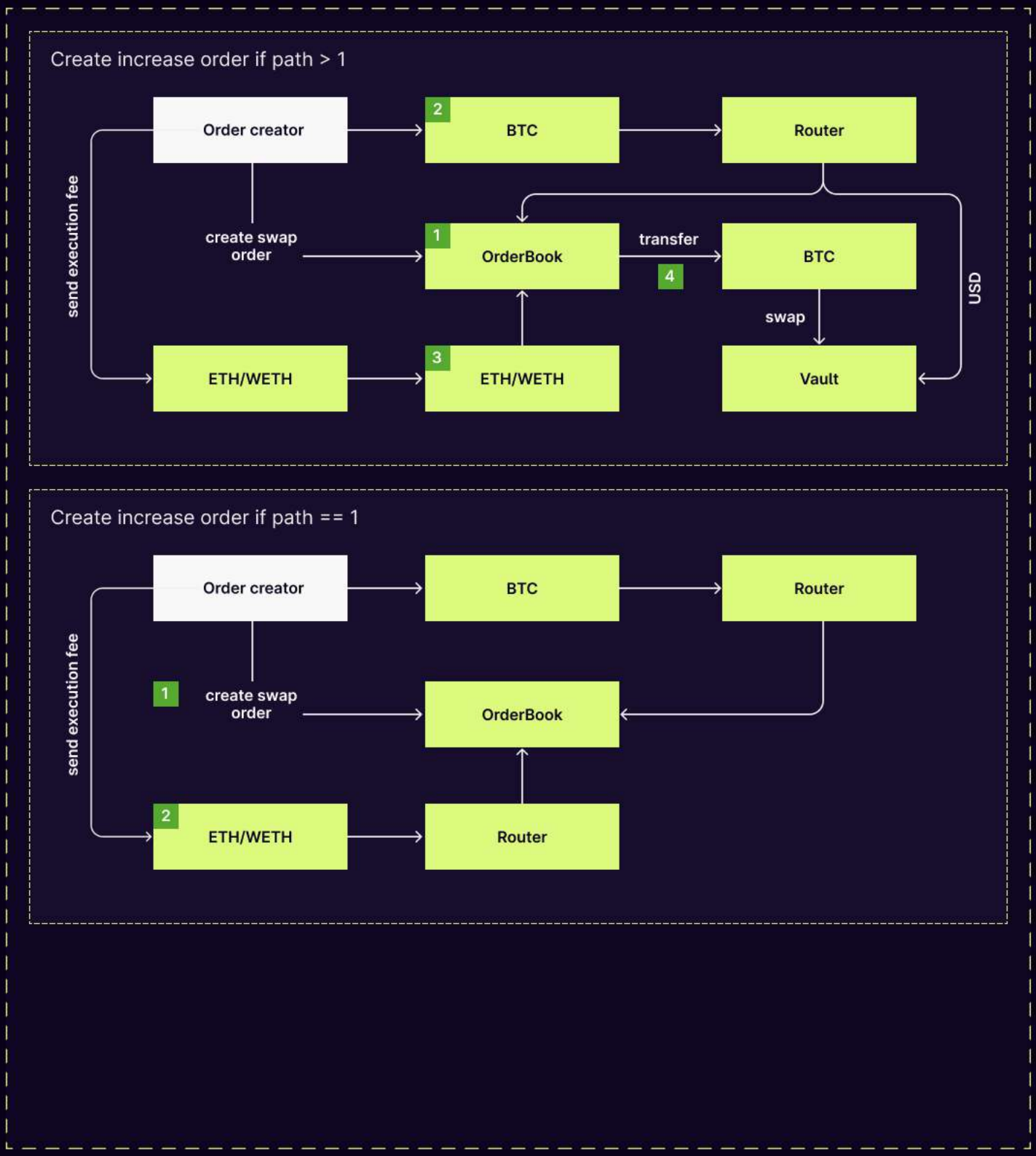


POSITIONMANAGER.SOL

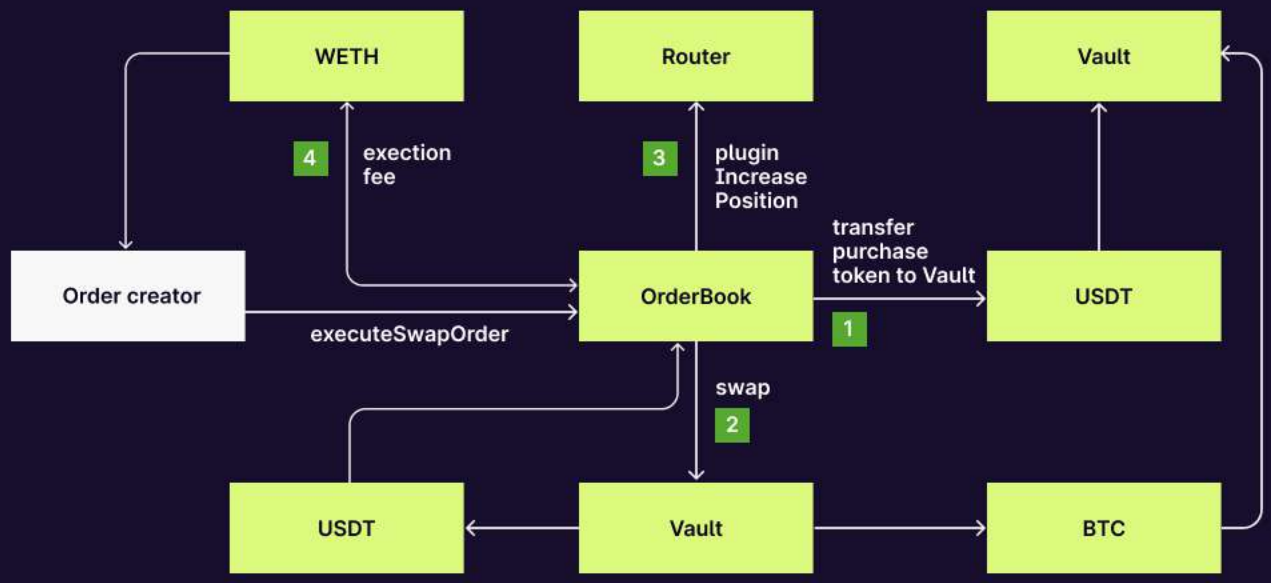
Increace/decreace position flow.



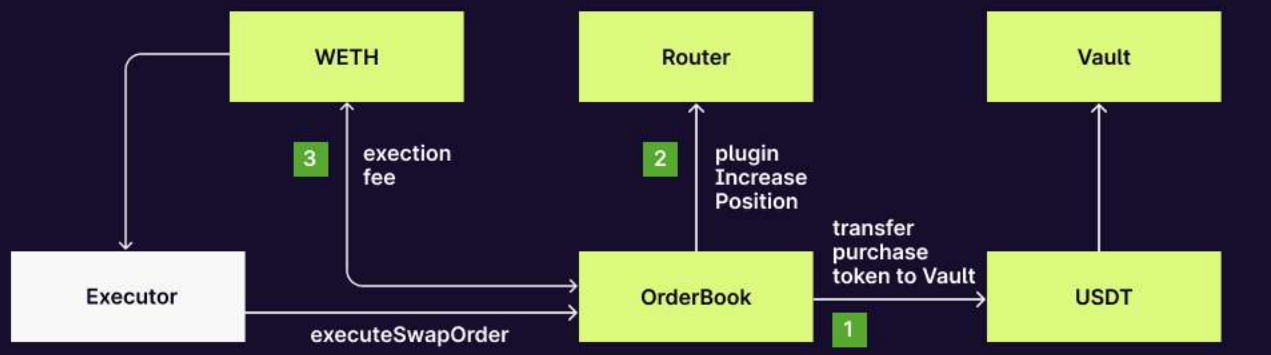

Create swap order btc usdt
Order creator
2
BTC
Router
create swap order
1
OrderBook
send execution fee
3
ETH/WETH
Router
Execute swap order
Executor
Execute SwapOrder
1
OrderBook
Transfer order amount to Vault
BTC
BTC
Execution Fee
2
Vault Swap
Vault
Order creator
3
USDT
Cancel Swap Order
return order body
1
Order creator
return execution fee
cancel swap order
BTC
OrderBook
WETH
Update Swap Order
Update minOut, triggerRatio, triggerAboveThreshold
Order creator
update swap order
OrderBook

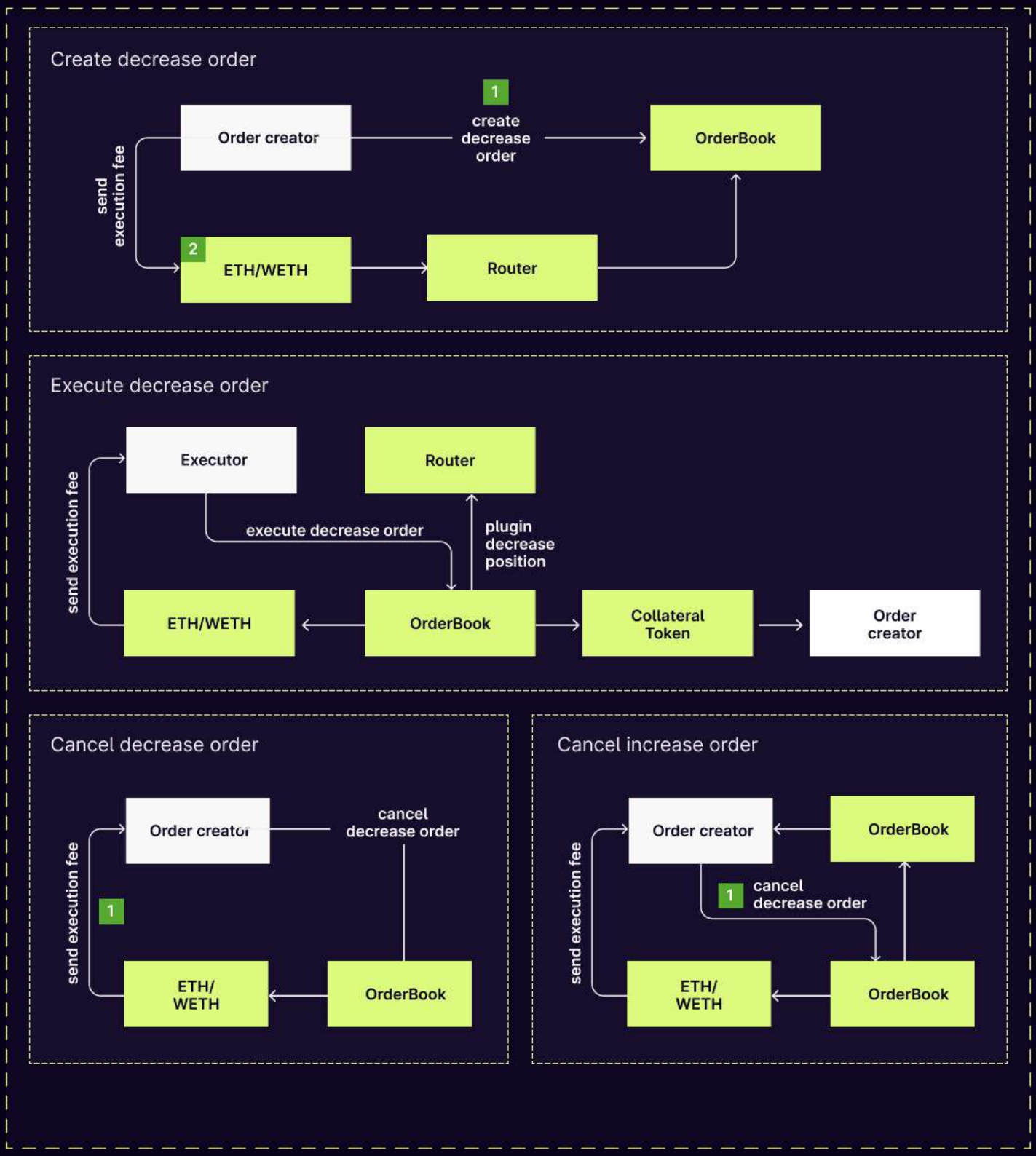

Create increase order if path > 1
Order creator
2
BTC
Router
send execution fee
create swap order
1
OrderBook
transfer
4
BTC
USD
swap
ETH/WETH
3
ETH/WETH
Vault
Create increase order if path == 1
Order creator
BTC
Router
send execution fee
1
create swap order
OrderBook
2
ETH/WETH
Router

Execute increase order if purchaseToken != collateralToken



Execute increase order if purchaseToken == collateralToken




Create decrease order
1
create decrease order
Order creator
OrderBook
send execution fee
2
ETH/WETH
Router
Execute decrease order
Executor
Router
send execution fee
execute decrease order
plugin decrease position
ETH/WETH
OrderBook
Collateral Token
Order creator
Cancel decrease order
Order creator
cancel decrease order
send execution fee
1
ETH/ WETH
OrderBook
Cancel increase order
Order creator
OrderBook
send execution fee
1
cancel decrease order
ETH/ WETH
OrderBook

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

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

Medium

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

Low

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

Informational

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

COMPLETE ANALYSIS

HIGH-1 | RESOLVED

Governance can withdraw deposit tokens.

RewardTracker.sol: withdrawToken().
The governance account can withdraw any ERC20 token from the contract's balance, including the deposit tokens. As a result, in case the private key of the governance account is exploited, users' funds can be withdrawn directly from the contract. That's why it is recommended to validate that the provided `_token` is not a deposit token to exclude any centralization risk.

Recommendation:
Validate that `_token` is not one of the deposit tokens.

Post-audit.
The token is validated not to be a deposit token now.

HIGH-2 | RESOLVED

Handlers can unstake, claim rewards, and transfer reward tracker tokens of any users to arbitrary receivers.

1) RewardTracker.sol: transferFrom(), claimForAccount(), unstakeForAccount().
Accounts with the handler role can:
- Claim rewards and unstake funds on behalf of other users to an arbitrary receiver without asking for the user's permission.
- Transfer RewardTracker tokens owned by users without any approval.

2) DPLPManager.sol: removeLiquidityForAccount().
Accounts with the handler role can remove the liquidity of any `_account`, burn GLP tokens of '_account' without providing an approval of the account and transfer the underlying tokens to an arbitrary `_receiver` address.

This creates a dangerous backdoor that works against the users of the protocol.

Recommendation:
Remove this dangerous backdoor or provide another kind of transactions on behalf of users, e.g. using EIP-3009 or EIP-2612.

Post-audit.
All the dangerous backdoors that would grant handlers access to users' funds were removed except for the overridden transferFrom function that allows handlers to transfer RewardTracker tokens of any user. Though the DPEX team has verified that such functionality is not a centralization risk for their protocol, it should still be noted in the final report. In all other functions, the handler can still claim, unstake, and remove liquidity on the behalf of users, but the receiver of the asset must be the account whose funds are accessed.

HIGH-3 | RESOLVED

ETH might get stuck on the contract.

BasePositionManager.sol: _transferOutETHWithGasLimitIgnoreFail(), line 285.
ETH is transferred to a msg.sender using the .send() function. However, the usage of the .send() function is considered deprecated. Also, since the function is used to transfer ETH, which belongs to the users, sending it without checking the status of .send() might make ETH stuck on the Position Manager contract. Such a restriction might also prevent valid smart-contracts (Gnosis multisig wallets, for example) in calling the functions. Thus, it is recommended to transfer ETH with .call() and custom gas units value (which should be enough to execute fallback) and check the result of the transfer to revert on any unexpected errors that would get ETH stuck on the Position Manager's balance (for example, if the user specifies a custom amount of gas for the transaction and the transfer fails due to the "out of gas" error).

Recommendation:
Transfer ETH with .call() and mandatory check of the result of the transfer OR verify that only user accounts (EOA) are supposed to call this function.

Post-audit.
.send() was replaced with .call() without gas limitation and the status of .call() is checked to catch an exception. All the external functions, within which _transferOutETHWithGasLimitIgnoreFail() is called, utilize the "nonReentrant" modifier, thus, the usage of .call() is performed in a safe way.

MEDIUM-1 | RESOLVED

Users cannot unstake tokens that are no longer deposit tokens.

RewardTracker.sol: _unstake(), line 254.
In case a certain token is no longer marked as a deposit token with the setDepositToken() function, users will no longer be able to unstake it. This means that the funds will be blocked until the token is marked as a deposit token again. The issue is marked as medium since the governance can block the access to users' funds at any time.

Recommendation:
Allow users to unstake tokens that are no longer marked as deposit tokens.

Post-audit.
It is now checked that the provided token has once been a deposit token so that it can be unstaked later.

MEDIUM-2 | ACKNOWLEDGED

Transferred Reward Tracker tokens might not be unstaked. RewardTracker.sol: _transfer().

When Reward Tracker tokens are transferred, values from the `stakedAmounts` and `depositBalances` mappings are not updated for the sender and recipient. As a result, when the user tries to unstake tokens, they will either have an insufficient balance of RewardTracker tokens or will not have enough value in `stakedAmounts` and `depositBalances`. Either way, the funds will be blocked until RewardTracker tokens are transferred back.

Recommendation:
Update the values in the `stakedAmounts` and `depositBalances` mappings so that the funds can be unstaked even after the transfers of RewardTracker tokens.

Post-audit.
According to the team, RewardTracker may grow into having multiple deposit tokens, thus iterating through tokens might increase gas expenditure and require additional security checks like using a nonReentrant modifier. Since the issue is not considered to be major, the current implementation won't be changed. All users of the dApp will be notified that they should be careful when transferring RewardTracker tokens as it may lead to the loss of the underlying tokens.

MEDIUM-3 | ACKNOWLEDGED

Governance can withdraw tokens from the Vault.

Vault.sol: upgradeVault().
Governance can transfer all the funds to an arbitrary address during the upgrades of the vault. Thus, in case the private key of Governance is exploited, funds can be stolen. Even if governance is a multisig account, it is still recommended to add extra validations of the `_newVault` parameter. In this case, it is recommended to check that `_newVault` supports the interface of Vault.sol.

Recommendation:
Validate that `_newVault` supports the interface of Vault.sol, for example, by using ERC-165 by OpenZeppelin.

Post-audit.
It is now validated that `_newVault` supports the interface of a previous Vault.sol. Therefore, it should be noted that in case the interface of the new vault contract is changed, it should be implemented that it supports the interface of the previous Vault smart contract.

Post-audit.
Due to the size limit, interface check was removed in commit 320887ccf885cc1a9b1a058df113c65790715125. However, The DPEX team has taken a responsibility to deploy a Timelock smart-contract ot ensure the safety of Governance actions.

MEDIUM-4 | RESOLVED

Transfer is not validated.

DPLPManager.sol: function _removeLiquidity(), line 248.
The transfer of USDG is performed with a regular transfer() method from the OpenZeppeling IERC20 interface without validating if the 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 the safeTransfer() function.

Recommendation:
Use SafeERC20 library.

Post-audit.
SafeERC20 is used now.

MEDIUM-5 | VERIFIED

Outdated Solidity version.

Currently, the protocol uses Solidity version 0.6.12, though the general security checklist recommends to utilize the newest stable version of Solidity, which is 0.8. The newest version of Solidity includes the latest optimization for the compiler, bug fixes, and features such as built-in overflow and underflow validations. Also, the contracts allow 0.6.0 compilation, which has major bugs in it. It is recommended to integrate the same version on all contracts.

Recommendation:
Update Solidity version to the latest version 0.8.

Post-audit.
The DPEX team has responded that 0.6.12 will be used. The team has assured that they will take care of the 0.6.12 features such as overflow/underflow.

MEDIUM-6 | RESOLVED

Wrong variable check.

DPLPManager.sol: function setShortsTrackerAveragePriceWeight(), line 80
The function requires a storage variable to be validated (shortsTrackerAveragePriceWeight) instead of the function argument (_shortsTrackerAveragePriceWeight). In this case, when an invalid value is given to the function, it will be impossible to change the variable again due to the invalid check. The issue is marked as medium since this variable is used in essential calculations.

Recommendation:
Check the function variable instead of a storage variable.

Post-audit.
The function argument is checked now.

LOW-1 | RESOLVED

Parameters lack validation.

Some crucial parameters that are set once take part in the essential logic such as fees calculations. The address that receives funds should be validated in contracts. Checking on zero addresses is necessary to avoid burn transfers and to ensure that the storage variables are set as intended. In case of the setGov() function, it could be possible to lose main admin functionality if allowed to pass zero address. Thus, it is recommended to add validations in the following functions:

1) RewardTracker.sol: initialize().

Validate that `_depositTokens[i]` is not added yet to avoid repeatable tokens adding and that it doesn't equal zero address.
Validate that `_distributor` is not equal to zero address.

2) Governable.sol: setGov().

Validate that `_gov` is not equal to zero address.

3) RewardRouter.sol: batchStakeGmxForAccount().

Validate that the length of the _accounts and _amounts arrays is the same.

4) Vault.sol: functions that include the `_receiver` parameter, function setGov().

5) BasePositionManager.sol: functions constructor(), setAdmin(), setReferralStorage().

6) RewardTracker.sol: function _claim().

7) OrderBook.sol: constructor(): all address parameters, setGov().

Recommendation:
Validate the parameters.

Post-audit.
In order to keep the size of Vault.sol below limit, validations were removed from it in commit 320887ccf885cc1a9b1a058df113c65790715125.

LOW-2 | RESOLVED

Decimals of Deposit tokens are not converted to common decimals. RewardTracker.sol: function _stake().

When tokens are staked, the amount is added to `stakedAmounts` (line 245),`totalSupply`, and `balances` (_mint(), lines 197-198). Thus, these values are summations of all staked deposit tokens. However, since deposit tokens might have different decimals, the summation won't reflect an actual sum of all deposit tokens, which can affect the rewards distribution. For example, user1 who has staked 1 token with 18 decimals, will receive more rewards than user2 who has staked 2 tokens with 6 decimals, despite the fact that user2 has staked a greater value.

Recommendation:
Verify that all deposit tokens will have the same amount of decimals or convert decimals of deposit tokens to common decimals.

Post-audit.
It is now validated during staking that the deposit token has only 18 decimals.

INFO-1 | RESOLVED

Gas optimization suggestions.

1) RewardTracker.sol: initialize().
`_depositTokens` should be marked as calldata instead of memory since the array isn't modified in the function.
Declaring the `depositToken` variable is unnecessary and increases gas expenditure, so the `_depositTokens[i]` value can be used directly.
2) Vault.sol: `hasDynamicFees`, `useSwapPricing`, `inManagerMode`, `inPrivateLiquidationMode`.
OrderBook.sol: `isInitialized`.
Assigning false to boolean storage variables is redundant and increases gas expenditure since boolean variables are initialized with false by default.

INFO-2 | RESOLVED

Lack of events.

Setters should emit events in order to keep track of the historical changes of storage variables. The following functions should emit events:
1) RewardTracker.sol: setDepositToken(), setInPrivateTransferMode(), setInPrivateStakingMode(), setInPrivateClaimingMode(), setHandler().
2) Governable.sol: setGov().
3) DPLPManager.sol: setInPrivateMode(), setShortsTracker(), setShortsTrackerAveragePriceWeight(), setHandler(), setCooldownDuration(), setAumAdjustment().
4) Vault.sol: all functions starting with “set” and functions clearTokenConfig(), addRouter(), removeRouter().

Recommendation.
Emit event in setters.

Post-audit.
In order to keep the size of Vault.sol below limit, events were removed from it in commit 320887ccf885cc1a9b1a058df113c65790715125.

INFO-3 | VERIFIED

Contracts look like they are upgradable but they are not.

RewardRouter.sol, Vault.sol, OrderBook.sol
The contracts have both the initialize() and constructor() functions. The initialize function implies that the contract is upgradable. However, based on the deploy scripts, the contracts are not upgradable, which is why it should be verified if the contracts are upgradable or not.

Recommendation:
Verify if the contracts should not be upgradable.

Post-audit.
According to the team, the contracts are not upgradable. Yet, the initialize method will be kept as all the deployment scripts are already designed to first deploy the contract and then call the initialize function. Also, it should be noted that only governance can call the initialize function, and it can only be called once so that initialization is performed in a secure way.

INFO-4 | RESOLVED

Execution of the increase/decrease order might revert.

OrderBook.sol: executeIncreaseOrder(), line 761; executeDecreaseOrder(), line 862.
When the increase/decrease position is executed, it is necessary for the address of the OrderBook to get approved as a plugin in Router.sol by the creator of the order. However, in case such approval is not granted, the execution of the order will revert, causing the executor to lose funds on gas without refunding. Thus, it is recommended to either provide a verification that OrderBook is an approved plugin for the creator of order on the dApp before an executor executes the order or validate this approval at the beginning of the function to minimize the losses of the executor.

Recommendation.
Validate that OrderBook is an approved plugin for the creator of the order either on the dApp or executions scripts before executing an order or at the beginning of the function.

Contracts are not compiling.

OrderBook.sol: IRouter.plugins is invoked as a variable, not a function. It should use () instead of []. In the IRouter interface, plugins should be marked as a view.
RewardTracker.sol: the commented claimForAccount and unstakeForAccount functions are still in the interface.
DPLPManager.sol: the commented removeLiquidityForAccount function is still in the interface.
IERC20.sol: the added decimals() function breaks login on every contract that uses this interface.

Recommendation:
Fix the contracts and check compiling.

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

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

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

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

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

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

	DPLPManager.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	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

Tests written by Zokyo Security

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

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

DPEX and DPLP

DPEX and DPLP initialization

✓ Should get correct Id (93ms)
✓ Should get correct gov
✓ Should get correct name and symbol (89ms)

RewardRouter

Gmx operations

✓ stakeGmxForAccount, stakeGmx, stakeEsGmx, unstakeGmx, unstakeEsGmx, claimEsGmx, claimFees, compound, batchCompoundForAccounts (4253ms)
✓ batchStakeGmxForAccount, claim, withdrawToken (1367ms)

Glp operations

✓ mintAndStakeGlp, unstakeAndRedeemGlp, compound, batchCompoundForAccounts (3684ms)
✓ mintAndStakeGlpEth, unstakeAndRedeemGlpEth, compoundForAccount (3078ms)

DPEX. OrderBook contract.

Test createSwapOrder function

✓ Should revert with 'OrderBook: invalid _path.length' (61ms)
✓ Should revert with 'OrderBook: invalid _path' (42ms)
✓ Should revert with 'OrderBook: invalid _amountIn'
✓ Should revert with 'OrderBook: insufficient execution fee' (40ms)
✓ Should revert with 'OrderBook: only weth could be wrapped' (43ms)
✓ Should revert with 'OrderBook: incorrect value transferred'
✓ Should revert with 'OrderBook: incorrect execution fee transferred'
✓ createSwapOrder, DAI -> BTC (73ms)
✓ Should emit CreateSwapOrder (56ms)
✓ Should emit CreateSwapOrder with 3 length of path array (69ms)
✓ Should swap orders index be equal 1 (73ms)

Test cancelSwapOrder function

✓ Should revert with 'OrderBook: non-existent order' (73ms)
✓ Should change alice balance after call with wrapped tokens (67ms)
✓ Should emit CancelSwapOrder (89ms)

Order manipulations

✓ Update swap order and execute (107ms)
✓ Revert if order is not exist
✓ Revert if minOut > amountOut (120ms)

Order Increase

✓ Create increase order
✓ Update increased order (53ms)
✓ Cancel increased order (60ms)
✓ Execute increased order (259ms)

Order Decrease

✓ Create decrease order
✓ Update decreased order (42ms)
✓ Cancel decreased order (40ms)
✓ Execute decreased order (61ms)

DPEX. OrderBookReader contract.

Test getSwapOrders function

✓ Should return data from first and second swap

Test getIncreaseOrders function

✓ Should return data from first and second increase Orders (151ms)

Test getDecreaseOrders function

✓ Should return data from first and second decrease Orders (67ms)

RewardTracker.sol

Test initialize function

✓ Should revert 'RewardTracker: already initialized'
✓ Should revert 'Governable: forbidden'

Test withdrawToken function

✓ Should change balance after call (56ms)
✓ Should revert 'Governable: forbidden' (51ms)

Test stake function

✓ Should revert with 'RewardTracker: action not enabled'
✓ Should revert 'Governable: forbidden'
✓ Should revert with 'RewardTracker: invalid _amount'
✓ Should revert with 'RewardTracker: invalid _depositToken'
✓ Should change contract balance after call (43ms)

Test stakeForAccount function

✓ Should change contract balance after call (61ms)
✓ Should revert 'Governable: forbidden'
✓ Should revert 'RewardTracker: mint to the zero address' (76ms)

Test unstake function

✓ Should change contract balance after call (82ms)
✓ Should revert with 'RewardTracker: action not enabled' (68ms)

Test unstakeForAccount function

✓ Should change contract balance after call (89ms)
✓ Should revert with 'RewardTracker: invalid amount' (73ms)

✓ Should revert with 'RewardTracker: invalid _depositToken' (65ms)
✓ Should revert with 'RewardTracker: _amount exceeds stakedAmount' (74ms)

Test transfer function

✓ Should change recipient balance after call (77ms)

Test transferFrom function

✓ Should change recipient balance after call (85ms)
✓ Should change recipient balance after call (81ms)
✓ Should revert with 'RewardTracker: transfer from the zero address' (78ms)
✓ Should revert with 'RewardTracker: transfer to the zero address' (81ms)
✓ Should revert with 'RewardTracker: forbidden' (84ms)

Test claim function

✓ Should change recipient balance after call (157ms)
✓ Should change recipient balance after call (90ms)
✓ Should revert with 'RewardTracker: action not enabled' (130ms)
✓ Should revert with 'RewardTracker: action not enabled' (155ms)

Test claimForAccount function

✓ Should change recipient balance after call (186ms)
✓ Should change recipient balance after call (95ms)
✓ Should revert with 'RewardTracker: action not enabled' (131ms)
✓ Should change recipient balance after call (137ms)

Test claimable function

✓ Should return 0 for staked amount (76ms)

Test setInPrivateClaimingMode function

✓ Should revert with 'Governable: forbidden'

Test setDepositToken function

✓ Should set deposit token to true
✓ Should revert with 'Governable: forbidden'

Test setInPrivateTransferMode function

✓ Should set set private transfer mode to true
✓ Should revert with 'Governable: forbidden'

Test allowance function

✓ Should set set private transfer mode to true

Test tokensPerInterval function

✓ Should return rewardDistributor tokens per interval;

Test Approve function

✓ Should revert with 'RewardTracker: approve to the zero address'

DPEX. Vault contract.

Test buyUSDG function

✓ Should revert 'Vault: _token not whitelisted' (40ms)
✓ Alice usdg balance should be equal 0, if receiver other address (161ms)
✓ Should change USDG bob balance (255ms)
✓ Reserve fee should be 1 (269ms)

✓ Amount arise usdg on vault after call (712ms)
✓ Pool amount should be 99 (258ms)
✓ Should revert 'Vault: forbidden'
✓ Should revert 'Vault: forbidden' after transfer (74ms)
✓ Should change balances after function call (288ms)
✓ Should change balances after call using min price (390ms)
✓ buyUSDG uses mintBurnFeeBasisPoints (366ms)
✓ Should revert with 'Vault: invalid tokenAmount' (146ms)
✓ buyUSDG adjusts for decimals (403ms)
✓ Should increase usdg amount (288ms)
✓ Should increase pool amount (285ms)
✓ Should mint usdg equal to mint amount (314ms)
✓ Should emit IncreaseUsdgAmount event (181ms)
✓ Should emit BuyUSDG event (199ms)

Test sellUSDG function

✓ Should transfer token out (406ms)
✓ Should burn amount (352ms)
✓ Should decrease pool amount (390ms)
✓ Should emit DecreasePoolAmount event (313ms)
✓ Should decrease usdg amount (382ms)
✓ Should emit DecreaseUsdgAmount event (362ms)
✓ Should emit SellUSDG event (327ms)

Test swap function

✓ Should increase usdg amount (715ms)
✓ Should decrease usdg amount (584ms)
✓ Should increase pool amount (551ms)
✓ Should decrease pool amount (540ms)
✓ Should transfer out amount (550ms)
✓ Should emit DecreasePoolAmount event (469ms)
✓ Should emit IncreasePoolAmount event (492ms)
✓ Should emit DecreaseUsdgAmount event (935ms)
✓ Should emit IncreaseUsdgAmount event (487ms)
✓ Should emit CollectSwapFees event (491ms)
✓ Should emit Swap event (470ms)

Test increasePosition function

✓ Should revert with error 'Vault: reserve exceeds pool' (653ms)
✓ Should increasePosition long & change data (498ms)
✓ Should revert with 'Vault: _size must be more than _collateral' (616ms)
✓ Should revert with 'Vault: reserve exceeds pool' (618ms)
✓ Should call increasePosition short & change data (783ms)
✓ Should call increasePosition short & change data in second time (967ms)
✓ Should emit IncreaseGuaranteedUsd event (586ms)
✓ Should emit CollectMarginFees event (636ms)

Test liquidatePosition function

✓ Should emit CollectMarginFees event (754ms)
✓ Should revert with 'Vault: empty position' (212ms)
✓ Should liquidate long position (945ms)
✓ Should revert 'Vault: invalid liquidator' (608ms)
✓ Should liquidate short position (1303ms)
✓ Should revert with 'Vault: empty position' (206ms)
✓ Should emit LiquidatePosition events (829ms)

Test decreasePosition function

✓ Should emit DecreasePosition & UpdatePosition events for long position (840ms)
✓ Should emit DecreaseGuaranteedUsd & DecreasePoolAmount events for long position (770ms)
✓ Should emit DecreasePosition & ClosePosition events for short position (738ms)

Test directPoolDeposit function

✓ Should emit IncreasePoolAmount (58ms)
✓ Should emit DirectPoolDeposit (62ms)
✓ Should emit DirectPoolDeposit (144ms)

Test withdrawFees function

✓ Should change balance after call (487ms)

Test updateCumulativeFundingRate function

✓ Should emit UpdateFundingRate event (218ms)

DPLPManager

Setters

✓ Private mode
✓ Short tracker
✓ Average price weight
✓ Handler
✓ Cooldown duration
✓ Aum adjustment

Getters

✓ Price (55ms)
✓ Aums (67ms)
✓ Global short average price

Main functionality

✓ Add liquidity (119ms)
✓ Add liquidity for account (113ms)
✓ Remove liquidity (202ms)
✓ Remove liquidity for user (198ms)

Revert

✓ When not gov try to set private mode
✓ When not gov try to set shorts tracker
✓ When not gov try to set shorts tracker average price weigh

- ✓ When try to set big shorts tracker average price weight
- ✓ When not gov try to set handler
- ✓ When not gov try to set cooldown duration
- ✓ When try to set big cooldown duration
- ✓ When not gov try to set aum adjustment
- ✓ When try to add liquidity in private mode
- ✓ When try to remove liquidity in private mode
- ✓ When not handler try to add liquidity for user
- ✓ When not handler try to remove liquidity for user
- ✓ When try to add liquidity with 0 amount
- ✓ When try to add liquidity with big minimum USDG (108ms)
- ✓ When try to add liquidity with big minimum GLP (88ms)
- ✓ When try to remove liquidity with 0 amount (102ms)
- ✓ When try to remove liquidity when cooldown is not passed (99ms)

Contract: BaseToken

- ✓ Correct deploy
- ✓ Gov should set new Gov
- ✓ Not Gov should not set new Gov
- ✓ Gov should set new token info
- ✓ Not Gov should not set new token info
- ✓ Gov should set yield tracker
- ✓ Not Gov should not set yield tracker
- ✓ Gov should add Admin
- ✓ Not Gov should not add Admin
- ✓ Gov should remove Admin
- ✓ Not Gov should not remove Admin
- ✓ Gov should withdraw randomly sent tokens (94ms)
- ✓ Not Gov should not withdraw randomly sent tokens (80ms)
- ✓ Gov should set InPrivateTransferMode
- ✓ Not Gov should not set InPrivateTransferMode
- ✓ Gov should set handler
- ✓ Not Gov should not set handler
- ✓ Admin should add NonStaking account
- ✓ Admin should not add added NonStaking account
- ✓ Not Admin should not add NonStaking account
- ✓ Admin should remove NonStaking account
- ✓ Admin should not remove non added NonStaking account
- ✓ Not Admin should non remove NonStaking account
- ✓ Admin should recover claim
- ✓ Not Admin should non recover claim
- ✓ User should claim
- ✓ User should view total staked amount

- ✓ User should view account staked amount (correct work if account not nonStakingAccounts)
- ✓ User should view account staked amount (correct work if account is nonStakingAccounts)
- ✓ User should transfer his tokens
- ✓ User should approve (and view allowance) transfer his tokens
- ✓ User should not approve transfer his tokens to ZERO_ADDRESS
- ✓ User should transfer from if hi is handler
- ✓ User should transfer from if hi is not handler
- ✓ User should not transfer from if hi is not handler and hi hasn`t allowance
- ✓ Minting is not possible to ZERO_ADDRESS (53ms)
- ✓ Minting correct work if recipient nonstaking accounts (85ms)
- ✓ Burning is not possible to ZERO_ADDRESS (49ms)
- ✓ Burning correct work if recipient nonstaking accounts (96ms)
- ✓ User should not transfer his tokens to ZERO_ADDRESS
- ✓ User should not transfer his tokens if inPrivateTransferMode activeted and he isn`t Handler
- ✓ User should transfer his tokens if inPrivateTransferMode activeted and he is Handler
- ✓ Transfer correct work if sender and recipient nonstaking accounts
- ✓ Updating rewards using yield tracker

Position Manager

- ✓ Check initialize
- ✓ setOrderKeeper
- ✓ setLiquidator
- ✓ setPartner
- ✓ setInLegacyMode
- ✓ setShouldValidateIncreaseOrder
- ✓ Increase Position: frobidden because in Legacy Mode and client is not a partner (47ms)
- ✓ PositionManager: invalid _path.length (48ms)
- ✓ BasePositionManager: mark price higher than limit (268ms)
- ✓ Increase Position Timelock: forbidden (263ms)
- ✓ Increase Position and Decrease Position (1731ms)
- ✓ increasePositionETH: frobidden because in Legacy Mode and client is not a partner
- ✓ increasePositionETH: Timelock: forbidden (68ms)
- ✓ increasePositionETH: PositionManager: invalid _path
- ✓ increasePositionETH: PositionManager: invalid _path.length (40ms)
- ✓ increasePositionETH: BasePositionManager: mark price higher than limit (125ms)
- ✓ increasePositionETH (288ms)
- ✓ increasePositionETH and decreasePositionETH (1191ms)
- ✓ increasePositionETH with swap (747ms)
- ✓ increasePosition and increasePositionETH to short (614ms)
- 1) decreasePositionAndSwap and decreasePositionAndSwapETH
- ✓ Execute Swap Order
- ✓ Execute Increase Order
- ✓ Execute Decrease Order
- ✓ Liquidate Position

Zokyo Security team has prepared their own set of unit-tests in order to check all the contracts within the scope. All the main logic, calculations, user flows and flow of funds were carefully checked during testing. Thus Zokyo Security team provided a sufficient tests-coverage for DPEX protocol.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the DPEX team

As a part of our work assisting DPEX in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the DPEX team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the DPEX team. All of them were also carefully checked by the team of auditors.

```
TokenManager
1)  "before each" hook for "inits"

GlpManager
2) "before each" hook for "inits"

OrderBook, cancelMultiple
✓ cancelMultiple (420ms)

OrderBook, decrease position orders
✓ Create decrase order, bad fee
createDecraseOrder gas used 296025
✓ Create decrease order, long (46ms)
updateDecreaseOrder gas used 52488
✓ updateDecreaseOrder (77ms)
createDecreaseOrder gas used 276113
✓ Create decrease order, short (49ms)
✓ Create two orders (67ms)
✓ Execute decrease order, invalid price (724ms)
✓ Execute decrease order, non-existent (44ms)
executeDecreaseOrder gas used 315759
✓ Execute decrease order, long (613ms)
executeDecreaseOrder gas used 322592
✓ Execute decrease order, short, BTC (328ms)
createSwapOrder 276089
executeDecreaseOrder gas used 336577
✓ Execute decrease order, long, BNB (403ms)
cancelDecreaseOrder gas used 87540
✓ Cancel decrease order (111ms)

OrderBook
✓ setGov (49ms)
✓ set*
✓ initialize, already initialized
```

OrderBook, increase position orders
✓ createIncreaseOrder, bad input (224ms)
✓ createIncreaseOrder, two orders (129ms)
createIncreaseOrder gas used 399400
✓ createIncreaseOrder, pay WETH (63ms)
createIncreaseOrder gas used 372538
✓ createIncreaseOrder, pay BNB (52ms)
createIncreaseOrder gas used 423776
✓ createIncreaseOrder, long A, transfer and purchase A (66ms)
createIncreaseOrder gas used 672714
✓ createIncreaseOrder, long A, transfer A, purchase B (153ms)
createIncreaseOrder gas used 403924
✓ createIncreaseOrder, short A, transfer B, purchase B (69ms)
createIncreaseOrder gas used 652814
✓ createIncreaseOrder, short A, transfer A, purchase B (139ms)
updateIncreaseOrder gas used 46841
✓ updateIncreaseOrder (88ms)
cancelIncreaseOrder gas used 105635
✓ cancelOrder (126ms)
cancelIncreaseOrder gas used 91973
✓ cancelOrder, pay BNB (103ms)
✓ executeOrder, non-existent order
✓ executeOrder, current price is invalid (1390ms)
executeIncreaseOrder gas used 483367
✓ executeOrder, long, purchase token same as collateral (239ms)
✓ executOrder, 2 orders with the same position (426ms)
executeIncreaseOrder gas used 655136
✓ executeOrder, long, swap purchase token to collateral (297ms)
executeIncreaseOrder gas used 510860
✓ executeOrder, short, purchase token same as collateral (224ms)
executeIncreaseOrder gas used 665340
✓ executeOrder, short, swap purchase token to collateral (283ms)
✓ executeOrder, short, pay BNB, no swap (274ms)
executeIncreaseOrder gas used 662840
✓ createIncreaseOrder, bad path

OrderBook, swap orders
✓ createSwapOrder, bad input (115ms)
createSwapOrder 349423
✓ createSwapOrder, DAI -> BTC (71ms)
createSwapOrder 325047
✓ createSwapOrder, WBNB -> DAI (90ms)

createSwapOrder 298161
✓ createSwapOrder, BNB -> DAI (90ms)
createSwapOrder 329535
✓ createSwapOrder, DAI -> WBNB, shouldUnwrap = false (42ms)
createSwapOrder 349435
createSwapOrder 298135
✓ createSwapOrder, two orders (112ms)
canceSwapOrder 102573
✓ cancelSwapOrder, tokenA != BNB (88ms)
canceSwapOrder 88872
✓ cancelSwapOrder, tokenA == BNB (78ms)
updateSwapOrder 54761
✓ updateSwapOrder (103ms)
executeSwapOrder 373629
✓ executeSwapOrder, triggerAboveThreshold == false (262ms)
executeSwapOrder 326281
✓ executeSwapOrder, triggerAboveThreshold == false, DAI -> WBNB, shouldUnwrap = false (152ms)
executeSwapOrder 375688
✓ executeSwapOrder, triggerAboveThreshold == true (223ms)
executeSwapOrder 507567
✓ executeSwapOrder, triggerAboveThreshold == true, BNB -> DAI -> BTC (319ms)
executeSwapOrder 362988
✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> BTC (413ms)
executeSwapOrder 498811
✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> DAI -> BTC (565ms)
executeSwapOrder 524272
✓ executeSwapOrder, triggerAboveThreshold == true, USDG -> BNB -> BTC (572ms)
executeSwapOrder 316003
✓ executeSwapOrder, triggerAboveThreshold == true, BTC -> USDG (301ms)
✓ complex scenario (612ms)

PositionManager core
3) "before each" hook for "inits"

PositionManager next short data calculations
4) "before each" hook for "PositionManager and GlpManager init with shortsTracker"

PositionRouter
✓ inits
✓ setAdmin (51ms)
✓ setDepositFee (89ms)
✓ setIncreasePositionBufferBps (49ms)
✓ setReferralStorage (50ms)
✓ setMaxGlobalSizes (103ms)
✓ withdrawFees (1278ms)
✓ approve (63ms)
✓ sendValue (44ms)
✓ setPositionKeeper (72ms)
✓ setMinExecutionFee (51ms)
✓ setIsLeverageEnabled (50ms)
✓ setDelayValues (72ms)
✓ setRequestKeysStartValues (61ms)
✓ increasePosition acceptablePrice long (360ms)
✓ increasePosition minOut long (372ms)
✓ validateExecution (778ms)
✓ validateCancellation (544ms)
✓ maxGlobalLongSize (754ms)
✓ decreasePosition acceptablePrice long (679ms)
✓ decreasePosition minOut long (969ms)
✓ increasePosition acceptablePrice short (607ms)
✓ maxGlobalShortSize (688ms)
✓ decreasePosition acceptablePrice short (560ms)
createIncreasePosition gas used 509423
executeIncreasePosition gas used 852582
cancelIncreasePosition gas used 142553
createIncreasePosition gas used 449469
executeIncreasePosition gas used 641272
✓ createIncreasePosition, executeIncreasePosition, cancelIncreasePosition (2137ms)
createIncreasePositionETH gas used 467076
executeIncreasePosition gas used 873659
cancelIncreasePosition gas used 131392
createIncreasePosition gas used 446947
executeIncreasePosition gas used 598406
✓ createIncreasePositionETH, executeIncreasePosition, cancelIncreasePosition (2629ms)
createIncreasePosition gas used 509423
executeIncreasePosition gas used 852582
createDecreasePosition gas used 393492
executeDecreasePosition gas used 481164
executeDecreasePosition gas used 646116

✓ createIncreasePosition, createDecreasePosition, executeDecreasePosition, cancelDecreasePosition (3368ms)
✓ executeIncreasePositions, executeDecreasePositions (7574ms)
✓ does not fail if transfer out eth fails (376ms)
✓ callback works (3704ms)
✓ invalid callback is handled correctly (1027ms)
Updates short tracker data
5) "before each" hook for "executeIncreasePosition"

Router
✓ setGov (53ms)
✓ addPlugin (49ms)
✓ removePlugin (84ms)
✓ approvePlugin
✓ denyPlugin (57ms)
✓ pluginTransfer (125ms)
✓ pluginIncreasePosition (123ms)
✓ pluginDecreasePosition (97ms)

buyUSDG gas used 357270
✓ swap, buy USDG (225ms)
sellUSDG gas used 357270
✓ swap, sell USDG (507ms)
swap gas used 361859
✓ swap, path.length == 2 (430ms)
swap gas used 403655
✓ swap, path.length == 3 (1057ms)
increasePosition gas used 684112
✓ swap, increasePosition (960ms)
✓ decreasePositionAndSwap (869ms)
✓ decreasePositionAndSwapETH (1038ms)

ShortsTracker
✓ inits
✓ getNextGlobalAveragePrice (208ms)
✓ setIsGlobalShortDataReady (65ms)
✓ setInitData (55ms)

Vault.averagePrice
6) "before each" hook for "position.averagePrice, buyPrice != markPrice"

Vault.buyUSDG
7) "before each" hook for "buyUSDG"

Vault.closeLongPosition
decreasePosition gas used 230989
✓ close long position (724ms)
decreasePosition gas used 231091
✓ close long position with loss (731ms)

Vault.closeShortPosition
decreasePosition gas used 234283
✓ close short position (602ms)
decreasePosition gas used 235420
✓ close short position with loss (619ms)

Vault.decreaseLongPosition
8) "before each" hook for "decreasePosition long"

Vault.decreaseShortPosition
9) "before each" hook for "decreasePosition short"

Vault.depositCollateral
10) "before each" hook for "deposit collateral"

Vault.settings
✓ directPoolDeposit (147ms)

Vault.fundingRates
decreasePosition gas used 315493
withdraw collateral gas used 319601
✓ funding rate (1576ms)

Vault.getFeeBasisPoints
✓ getFeeBasisPoints (1052ms)

Vault.getPrice
✓ getPrice (369ms)
✓ includes AMM price (605ms)

Vault.increaseLongPosition
11) "before each" hook for "increasePosition long validations"

Vault.increaseShortPosition
12) "before each" hook for "increasePosition short validations"

Vault.liquidateLongPosition
13) "before each" hook for "liquidate long"

Vault.liquidateShortPosition
14) "before each" hook for "liquidate short"

Vault.sellUSDG
15) "before each" hook for "sellUSDG"

Vault.settings
✓ inits (49ms)
✓ setVaultUtils (58ms)
✓ setMaxGlobalShortSize (91ms)
✓ setInManagerMode (61ms)
✓ setManager (61ms)
✓ setInPrivateLiquidationMode (63ms)
✓ setIsSwapEnabled (61ms)
✓ setIsLeverageEnabled (63ms)
✓ setMaxGasPrice (56ms)
✓ setGov (64ms)
✓ setPriceFeed (59ms)
✓ setMaxLeverage (92ms)

✓ setBufferAmount (64ms)
✓ setFees (196ms)
✓ setFundingRate (188ms)
✓ setTokenConfig (443ms)
✓ clearTokenConfig (302ms)
✓ addRouter
✓ removeRouter (67ms)
✓ setUsdgAmount (255ms)
✓ upgradeVault (82ms)
✓ setErrorController (180ms)

Vault.swap
16) "before each" hook for "swap"

Vault.withdrawCollateral
17) "before each" hook for "withdraw collateral"

Vault.withdrawFees
✓ withdrawFees (656ms)
✓ withdrawFees using timelock (763ms)
✓ batchWithdrawFees using timelock (849ms)

GMT
✓ inits
✓ setGov (124ms)
✓ addAdmin (63ms)
✓ removeAdmin (82ms)
✓ setNextMigrationTime (112ms)
✓ beginMigration (226ms)
✓ addBlockedRecipient (63ms)
✓ removeBlockedRecipient (76ms)
✓ addMsgSender (59ms)
✓ removeMsgSender (81ms)
✓ withdrawToken (187ms)
✓ transfer (97ms)
✓ approve
✓ transferFrom (144ms)
✓ allows migrations (217ms)

Treasury

✓ initialize (96ms)
✓ setGov (63ms)
✓ setFund (61ms)
✓ extendUnlockTime (45ms)
✓ addWhitelists (69ms)
✓ removeWhitelists (105ms)
✓ updateWhitelist (110ms)
✓ swap (356ms)
✓ validates swap.busdSlotCap (301ms)
✓ validates swap.busdHardCap (206ms)
✓ validates swap.isSwapActive (118ms)
✓ addLiquidity (228ms)
✓ withdrawToken (74ms)
✓ increaseBusdBasisPoints (56ms)
✓ endSwap (48ms)

FastPriceFeed

✓ inits (73ms)
✓ setSigner (93ms)
✓ setUpdater (72ms)
✓ setFastPriceEvents (75ms)
✓ setVaultPriceFeed (63ms)
✓ setMaxTimeDeviation (62ms)
✓ setPriceDuration (82ms)
✓ setMaxPriceUpdateDelay (75ms)
✓ setSpreadBasisPointsIfInactive (76ms)
✓ setSpreadBasisPointsIfChainError (63ms)
✓ setMinBlockInterval (68ms)
✓ setIsSpreadEnabled (54ms)
✓ setTokenManager (42ms)
✓ setMaxDeviationBasisPoints (40ms)
✓ setMaxCumulativeDeltaDiffs (55ms)
✓ setPriceDataInterval (53ms)
✓ setMinAuthorizations (42ms)
✓ setLastUpdatedAt (45ms)
✓ setPrices (162ms)
✓ favorFastPrice (226ms)
✓ getPrice (456ms)
✓ setTokens (159ms)
✓ setCompactedPrices (975ms)

tx0 setPricesWithBits gas used 146283
tx1 setPricesWithBits gas used 222781
✓ setPricesWithBits (1019ms)
tx0 setPrices gas used 108746
tx1 setPrices gas used 94276
tx2 setPrices gas used 83475
tx3 setPrices gas used 85681
✓ price data check (580ms)

BatchSender
✓ setHandler (178ms)

GmxTimelock
✓ inits (125ms)
✓ setTokenConfig (285ms)
✓ setBuffer (129ms)
✓ setIsAmmEnabled (49ms)
✓ setMaxStrictPriceDeviation (46ms)
✓ setPriceSampleSpace (51ms)
✓ setVaultUtils (60ms)
✓ setIsSwapEnabled (50ms)
✓ setContractHandler (49ms)
✓ setIsLeverageEnabled (164ms)
✓ setMaxGlobalShortSize (55ms)
✓ setMaxGasPrice (51ms)
✓ setMaxLeverage (70ms)
✓ setFundingRate (106ms)
✓ transferIn (109ms)
✓ approve (459ms)
18) processMint
✓ setGov (472ms)
✓ setPriceFeed (282ms)
19) withdrawToken
✓ vaultSetTokenConfig (319ms)
✓ priceFeedSetTokenConfig (324ms)
✓ addPlugin (279ms)
20) addExcludedToken
21) setInPrivateTransferMode
✓ setAdmin
✓ setExternalAdmin (159ms)

✓ setInPrivateLiquidationMode (73ms)
✓ setLiquidator (192ms)
✓ redeemUsdg (728ms)

OrderBookReader
✓ getIncreaseOrders (122ms)
✓ getDecreaseOrders (71ms)
✓ getSwapOrders (109ms)

PriceFeedTimelock
22) "before each" hook for "inits"

Reader
✓ getVaultTokenInfo (164ms)

ShortsTracker
✓ inits
✓ setBuffer (47ms)
✓ setAdmin (81ms)
✓ setHandler (43ms)
✓ setGov (86ms)
✓ setAveragePriceUpdateDelay (72ms)
✓ setMaxAveragePriceChange (98ms)
✓ setIsGlobalShortDataReady (83ms)
✓ disableIsGlobalShortDataReady (93ms)
✓ setGlobalShortAveragePrices (265ms)

Timelock
23) "before each" hook for "inits"

ReferralStorage
✓ Sets new handler (112ms)
✓ setTier (57ms)
✓ setReferrerTier
✓ setReferrerDiscountShare (41ms)
✓ setTraderReferralCode (66ms)

✓ Registers code (81ms)
✓ setCodeOwner (78ms)
✓ govSetCodeOwner (80ms)
✓ getTraderReferralInfo (56ms)
✓ timelock.setTier (73ms)
✓ timelock.setReferrerTier (76ms)
✓ timelock.govSetCodeOwner (67ms)

BonusDistributor
24) "before each" hook for "distributes bonus"

RewardRouter
25) "before each" hook for "inits"

RewardRouterV2
26) "before each" hook for "inits"

RewardTracker
27) "before each" hook for "inits"

Vester
28) "before each" hook for "inits"

Bridge
29) "before each" hook for "wrap, unwrap"

TimeDistributor
✓ distribute (335ms)

USDG
✓ addVault (86ms)
✓ removeVault (137ms)
✓ mint (154ms)
✓ burn (156ms)

YieldFarm
✓ stake (72ms)
✓ unstake (88ms)

YieldToken
tranfer0 gas used 60894
tranfer1 gas used 128552
tranfer2 gas used 275132
tranfer3 gas used 129710
✓ claim (536ms)
✓ nonStakingAccounts (621ms)

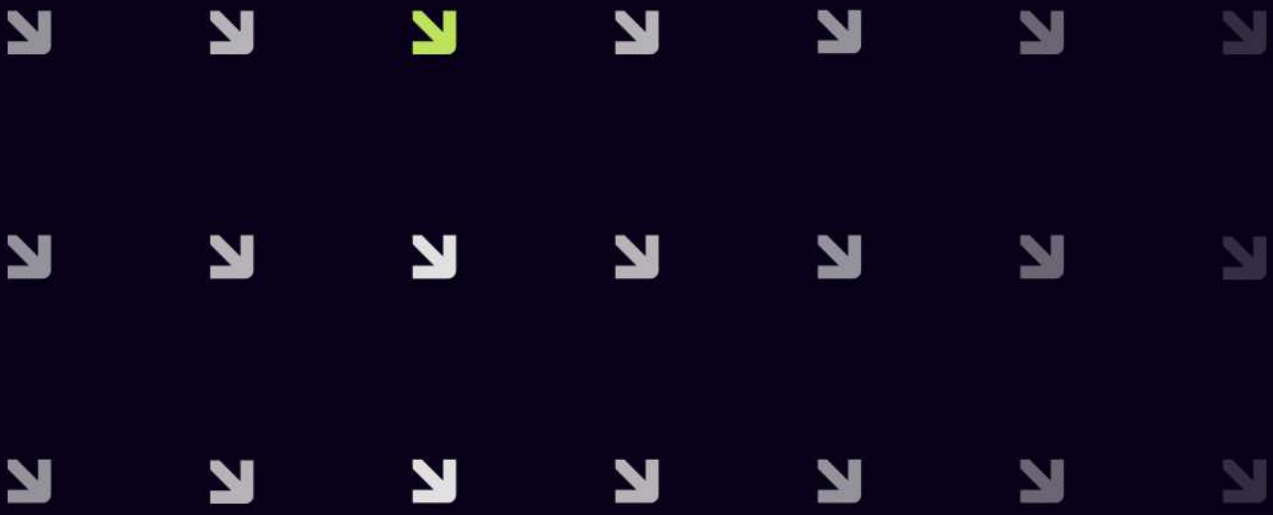
254 passing (5m)
29 failing

FILE	% STMTS	% BRANCH	% FUNCS
DPEX.sol	0	100	0
DPLP.sol	0	100	0
BaseToken.sol	0	0	0
MintableBaseToken.sol	0	0	0
OrderBookReader.sol	100	100	100
OrderBook.sol	100	85.71	100
PositionManager.sol	0	0	0
BasePositionManager.sol	89.13	89.39	100
Governable.sol	100	100	100
RewardTracker.sol	0	0	3.13
RewardRouter.sol	0	0	0
Vault.sol	84.89	67.72	94.19
DPLPManager.sol	0	0	0
All files	36.46	49.44	30.56

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

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

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



zokyo