LendHub – DeFi Lending and Borrowing

LLD Documentation

Document Control

Version	Date	Author	Description
1.1	26 – March- 2023	Shubham Prajapati	Introduction, Architecture and Unit test cases defined.
1.2	27 – March- 2023	Sasibhushan Upadrasta	Updates and modifications, added graph,UML, etc.
1.3	27 – March- 2023	Shubham Prajapati	Add the Testcases and Class diagram SVG image.

Approval Status

Version	Review Date	Reviewed By	Approved By	Comments

Table of Contents

1. INTRODUCTION

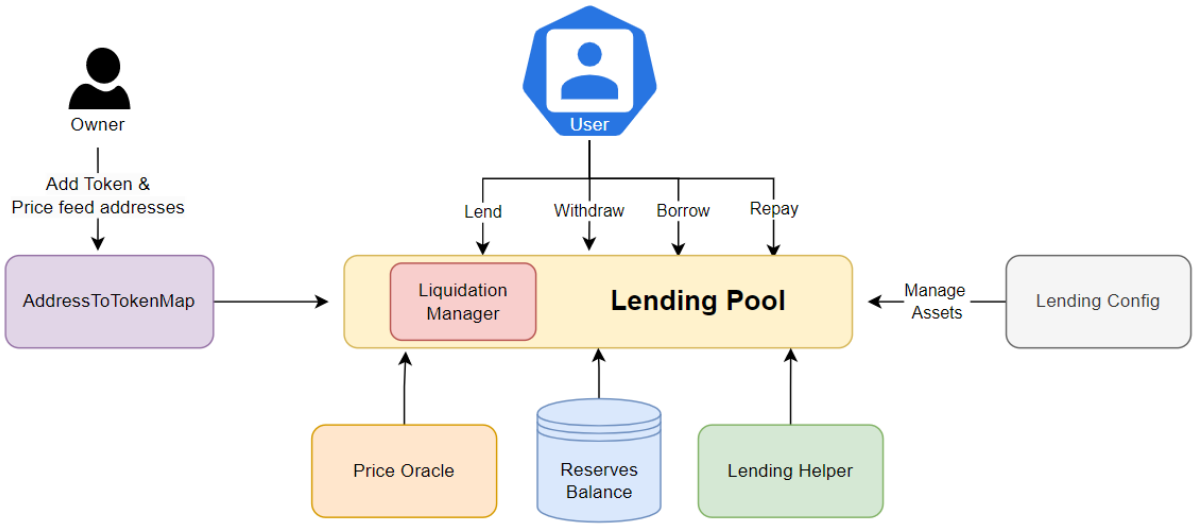
1.1 Why this -Level Design Document?

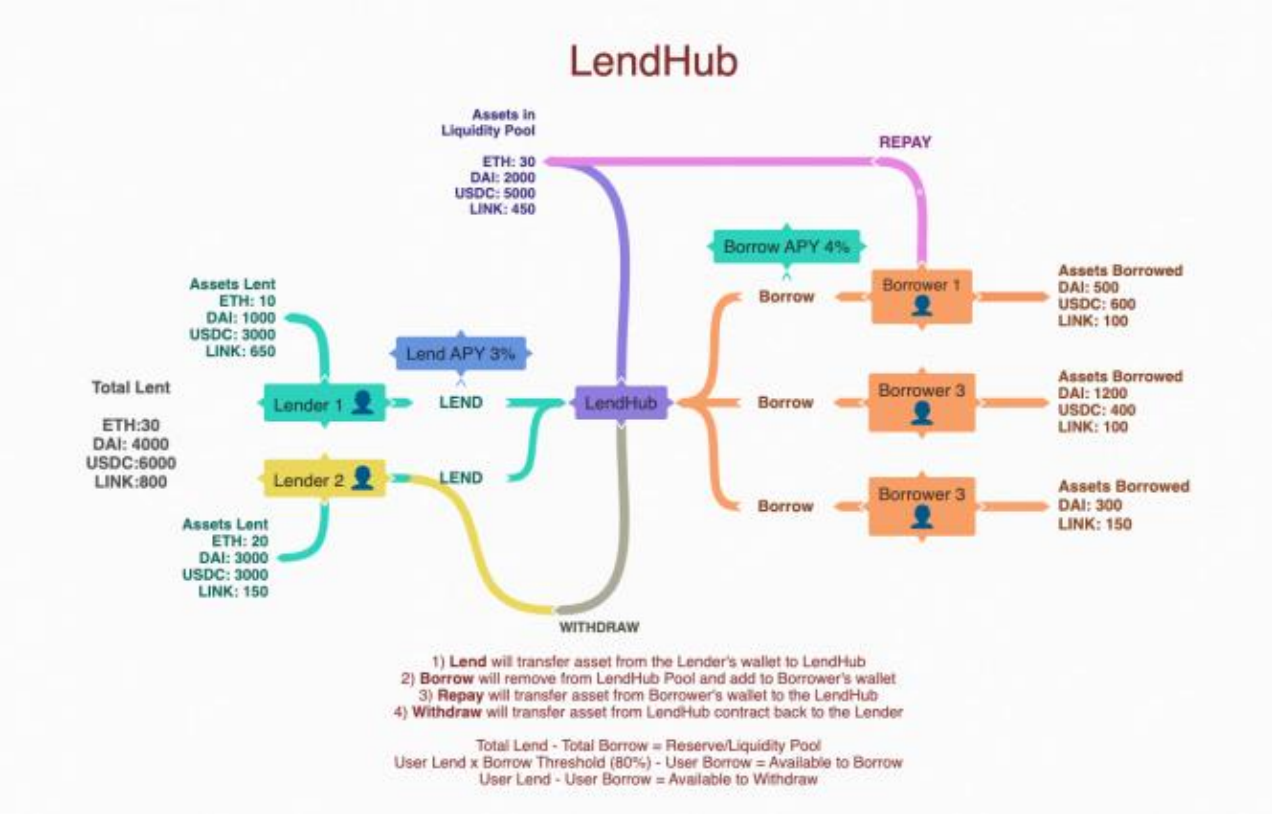
The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Lendhub Decentralized Finance App. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2 Scope

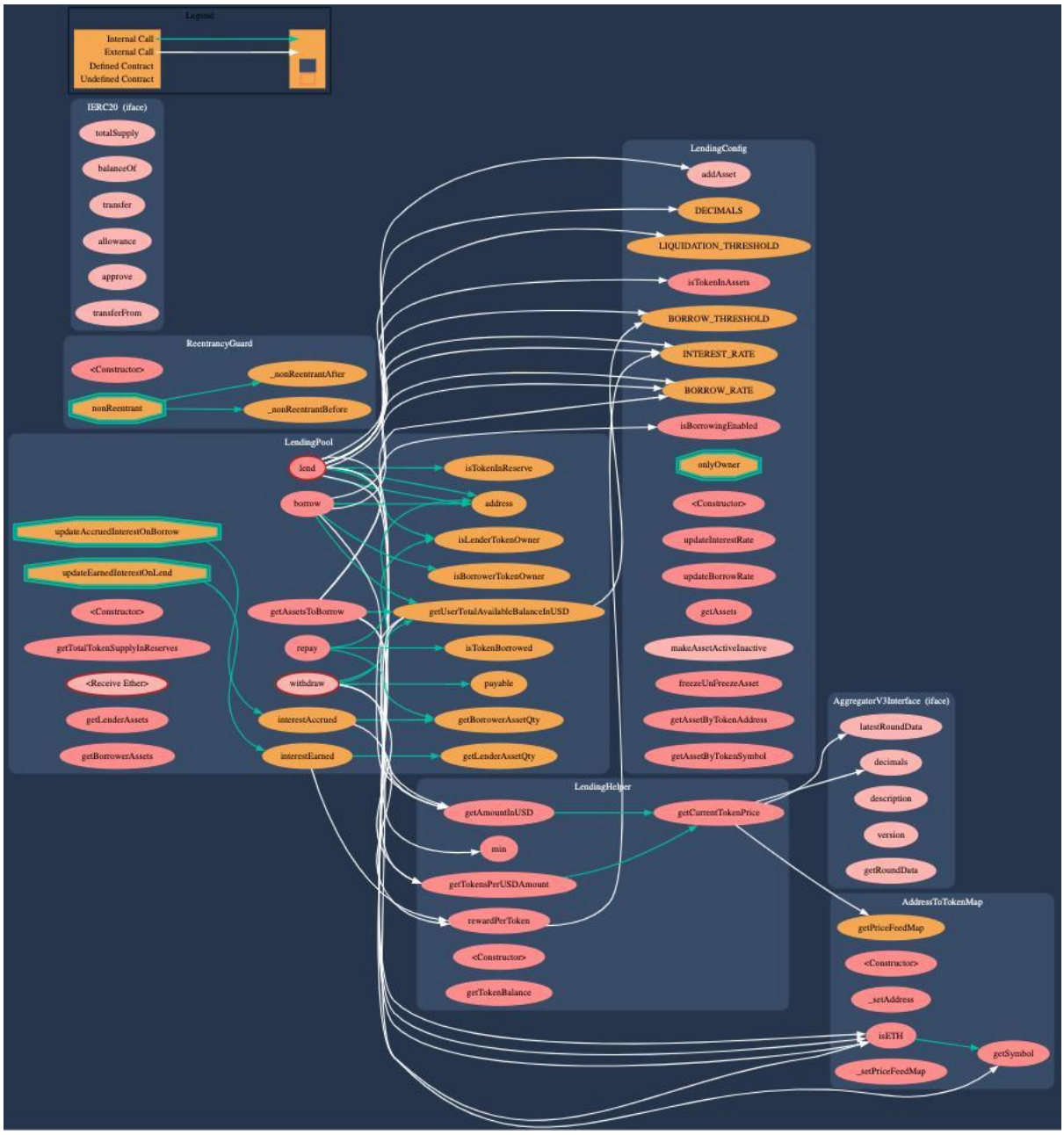
Low-level design (LLD) is a component-level design process that follows a step-by- step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work

2. Architecture

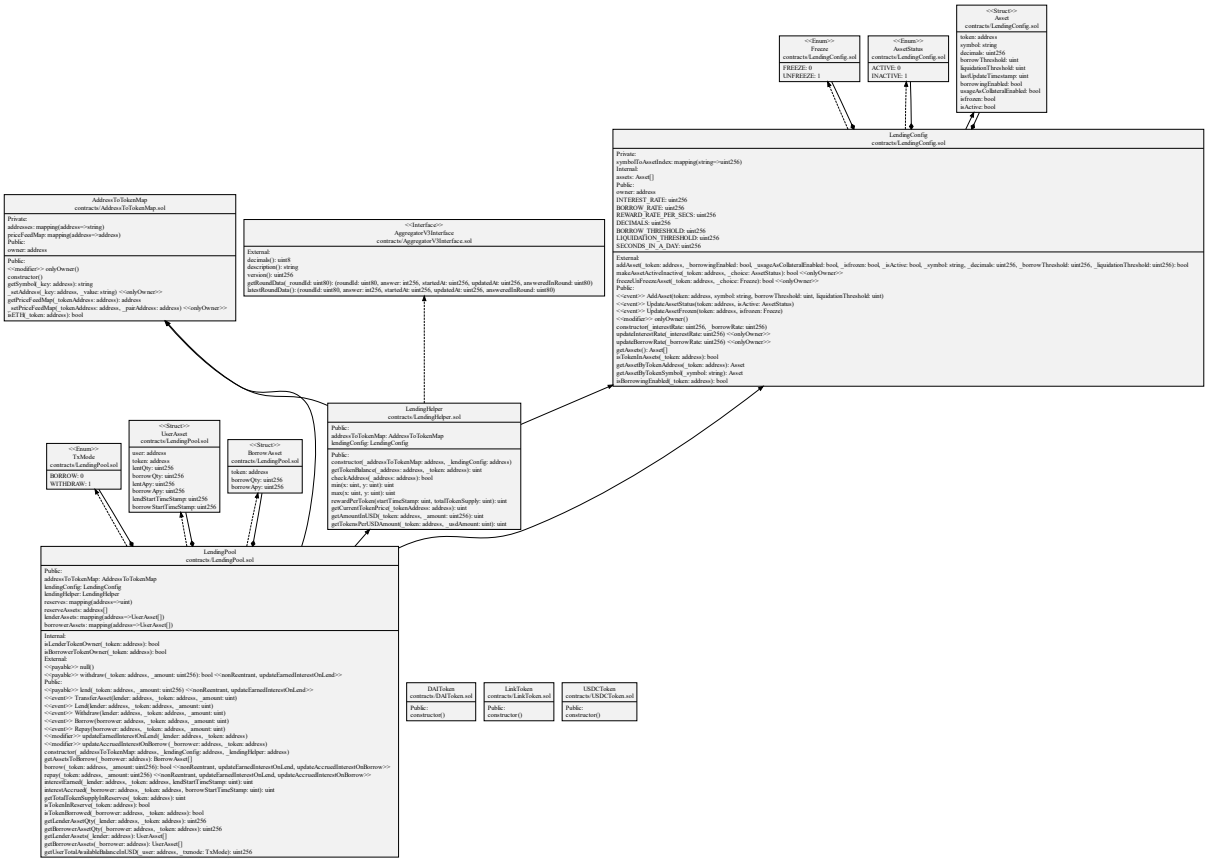




Contract Connectivity Graphs - Helpful in understanding the flow of information and for smart contract audit



UML – Class Diagram



3. Architecture Description

3.1 AddressToTokenMap

This contract is responsible for maintaining mapping between token address (ERC20 address) to symbols and Token to Chainlink price feed address mapping (address that can be used to retrieve token USD price from Chainlink oracles).

3.2 LendingConfig

This contract does the configuration of Assets and maintains them. It sets a lent interest rate and borrow interest rate. As they are dynamic in nature and are based on the volatility of the assets, hence subjected to change, there are some maintenance functions as well. It also includes some helper functions for the upkeep of the assets.

3.3 LendingHelper

This contract can be construed as a library rather than a contract. It has token related functions to help with building the core logic. It also has a function to calculate the logic to calculate the reward per token lent or interest accrued per borrowed.

3.4 LendingPool

This contract has functions that run core logic of the DeFi App.

- The lend functionality allows users to lend tokens to the DeFi platform to obtain interest rates better than conventional rates for fiat currency. Using this platform for digital asset storage will yield greater returns and also provide stability to the token. Some examples of coins/tokens that can be lent are ETH, LINK, MATIC, and stable coins like USDC and DAI.
- The lent assets and borrowed assets show the user that have been lent and borrowed respectively
- The borrow functionality lets the user borrow any asset available in the reserve/liquidity pool for short term loan at a borrow APY that is higher than the lent app. The difference in APY is made by the DeFi App.
- The repay functionality lets the user repay borrowed assets plus the borrowed interest accrued to the DeFi App.
- The withdraw functionality lets the user withdraw assets quantities that are lent minus borrowed

3.5 Price Oracle

- This is a Chainlink API which retrieves the current price of the coin/token using a price oracle called AggregatorV3.

3.5 Reserve Pool

- This has the DeFi App's liquidity or reserves any coin/token at any moment of time. This is very useful for controlling and determining the volatility of any asset.

4. Unit Test Cases

Test Case Description	Pre - Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1.Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1.Application URL is accessible 2.Application is deployed	The Application should load completely for the user when the URL is accessed
Verify whether the user is able to connect with the MetaMask wallet.	1.Metamask is installed in the chrome	The user should be able to connect with MetaMask wallet.
Verify whether the user is able to successfully connect with the MetaMask wallet.	1.Application is accessible. 2. User connected with the MetaMask wallet	The user should be able to successfully connect with MetaMask wallet.
Verify whether the user MetaMask wallet is able to connect with the Sepolia Testnet	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user MetaMask wallet should be able to successfully connect with Sepolia testnet.
Verify whether the user is able to see all the MetaMask assets (ETH /Tokens)	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to see all the MetaMask assets (ETH/Tokens)
Verify whether the user is able to enter only numerical values in the amount field of every modal.	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to enter only numerical values in the amount field of every modal.
Verify whether the user is able to Lend ETH / Token	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to lend ETH / Token

Verify whether the user is able to withdraw ETH / Tokens	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to withdraw ETH / Tokens
Verify whether the user is able to borrow tokens	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to borrow token
Verify whether the user is able to repay borrowed token	1.Application is accessible. 2. User connected with the MetaMask wallet. 3.Metamask is connect with Sepolia Testnet	The user should be able to repay borrowed Token

4.1 LendHub Smart Contract Testcases:

```
LendHub Tests
  √ 1. Should be able to retrieve all token symbols
  √ 2. Should be able retrieve all price feed
  √ 3. Lender Should be able to add assets
  √ 4. DAI Token should be in assets
  √ 5. USDC Token should not be in assets
  √ 6. Should be able to return symbol by passing address
  √ 7. Should be able to return address by passing symbol
  √ 8. Is Borrowing Enabled
  √ 9. Lender should not be able to make assets inactive
  √ 10. Only Deployer should be able to make assets inactive
  √ 11. Lender should not be able to freeze assets
  √ 12. Only Deployer should be able to freeze assets
  √ 13. Lender1 Should be able to lend 10 ETH
  √ 14. Lender1 lendQty Should be able to update with 10 more ETH
  √ 15. Lender1 Should be able to lend 1000 DAI
  √ 16. Not lend user should not be able to withdraw ETH
  √ 17. Not lend user should not be able to withdraw DAI
  √ 18. Lender1 Should be able to withdraw ETH assets
  √ 19. Lender1 Should be able to withdraw DAI
  √ 20. Lender2 Should be able to lend 5 ETH
  √ 21. Borrower2 Should be able to borrow 100 DAI
  √ 22. Borrower2 Should not be able to borrow more than 80% of his supplied
  √ 23. Borrower2 Should not be able to borrow more than reserve qty
  √ 24. Borrower2 Should be able to borrow 100 DAI again
  √ 25. Borrower2 Should be able to repay


25 passing (11s)
```