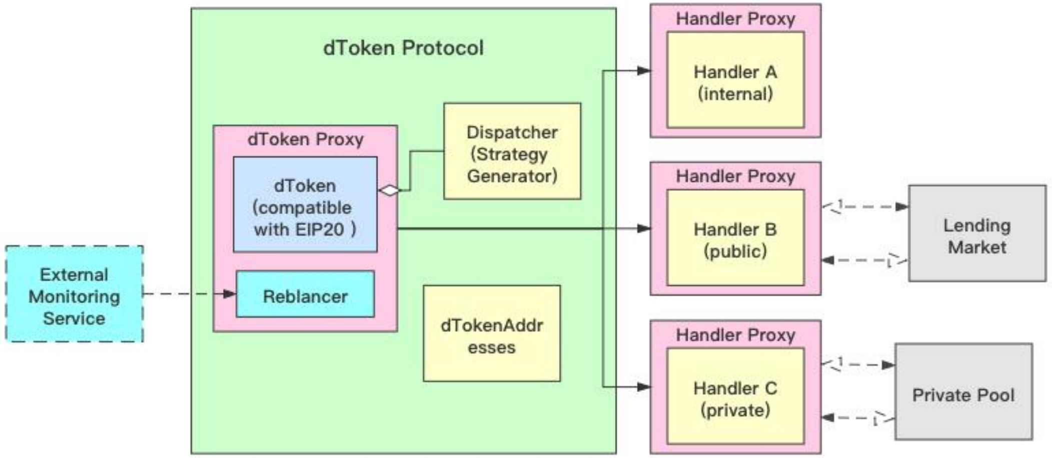
# dToken System Design

## Summary:

dToken is an interest-bearing aggregator, representing a pro rata claim of a specific ERC20 token plus yields earned. Users mint dToken to earn interest (i.e. deposit USDT and receive dUSDT in return). dToken can be transferred, traded, and redeemed at any time. dToken can attend to a variety of yielding protocols (i.e. Compound, Aave, dydx) and dForce PPP (Public-Private-Pool) (public pool for suppliers and private collateral pool accessible to verified borrowers only).

Users will receive dToken when they deposit a specific ERC20 token into the smart contract, which is redeemable at any time and represents a pro rata claim of corresponding ERC20 token plus interest earned.

## Architecture:



**Modules：**

- ✓ dTokenThe dToken core contract facilitates Mint/Redeem function, standard ERC20 interface and governance interface, and adjusts the proportion of tokens allocated to each lending market for optimum yield and liquidity.

- ✓ Dispatcher

  Attends to manage the Handler, Mint/Redeem strategy interfaces. In principle, dTokens with the same strategies can use the same Dispatcher. In practice, however, each dToken has an independent Dispatcher.

- ✓ dTokenAddresses

  Provide mappings of dToken and underlying token, all dTokens and Handlers share the same dTokenAddress contract.

- ✓ Handler

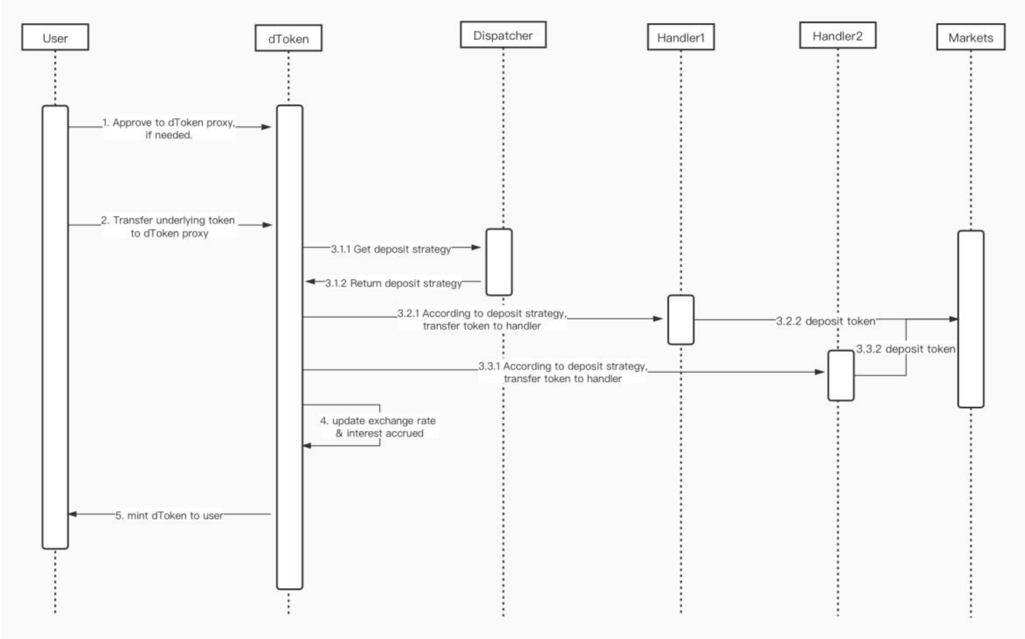
  Provide a unified interface to facilitate the wrapping of lending markets, i.e. deposit/withdraw, and query interfaces, etc. Internal Handler is the default Handler of dToken and cannot be removed. The Handler of Lending Market can support multiple dTokens simultaneously.

- ✓ External Monitoring Service（to be implemented）

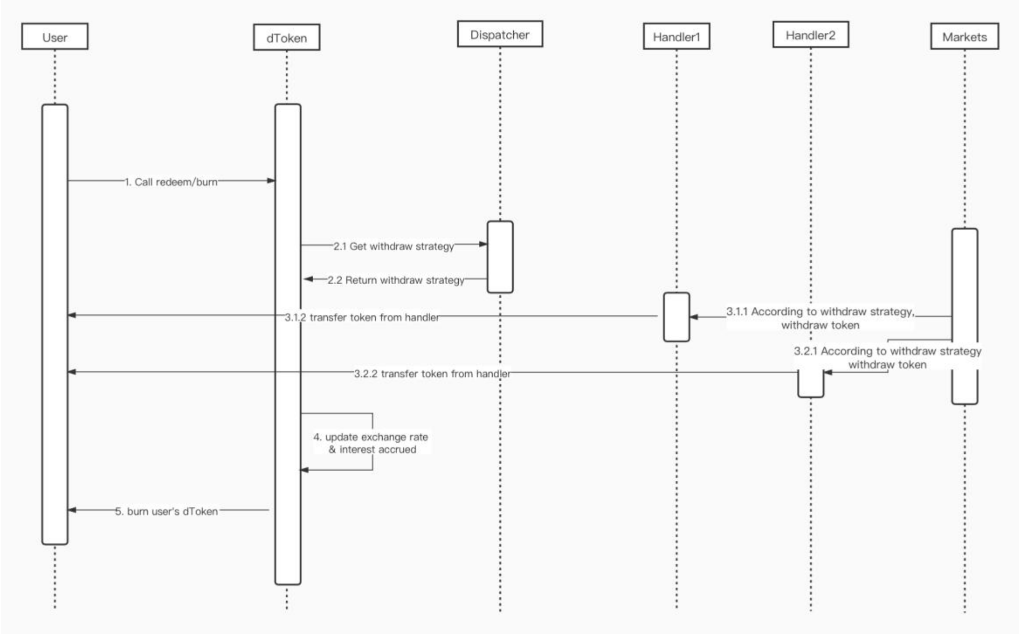
Implement basic strategy to trigger rebalance function externally, in quest of both optimum yield and the best liquidity.

**Key Features：**

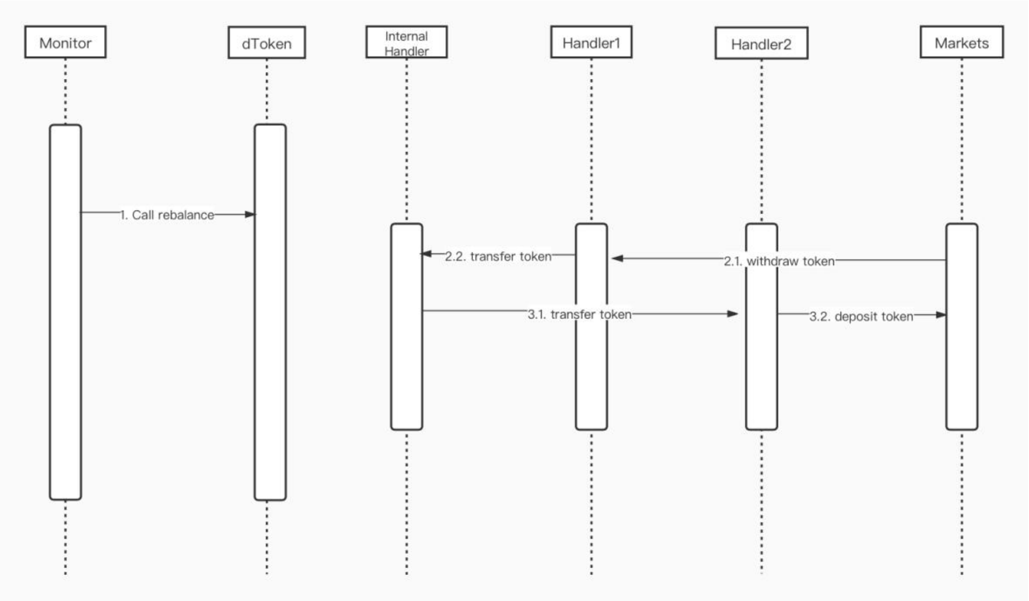
✓ Mint: the process to deposit underlying Token into a smart contract and trigger the automatic conversion of dToken. The underlying Token will be supplied to selected lending markets on a pro rata basis according to their shares of the whole.


User
dToken
Dispatcher
Handler1
Handler2
Markets
1. Approve to dToken proxy, if needed.
2. Transfer underlying token to dToken proxy
3.1.1 Get deposit strategy
3.1.2 Return deposit strategy
3.2.1 According to deposit strategy, transfer token to handler
3.2.2 deposit token
3.3.1 According to deposit strategy, transfer token to handler
3.3.2 deposit token
4. update exchange rate & interest accrued
5. mint dToken to user

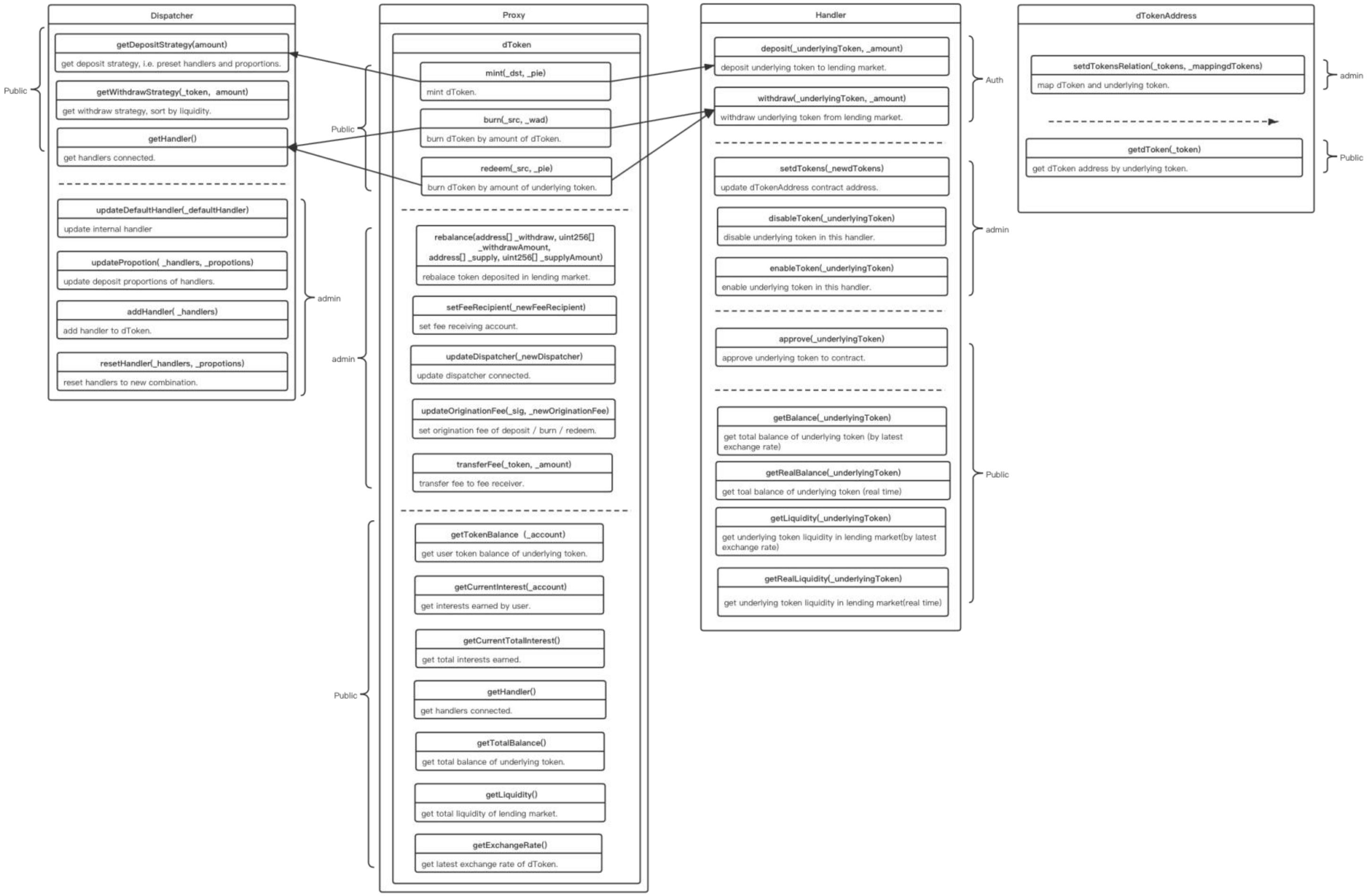
✓ Burn/Redeem: attends to burn/redeem dToken and withdraw the underlying Token, with balances sitting in the Internal Handler consumed first. In the event balances failed to fulfill the liquidity withdrawal, it will call Handlers in liquidity priority.


User
dToken
Dispatcher
Handler1
Handler2
Markets
1. Call redeem/burn
2.1 Get withdraw strategy
2.2 Return withdraw strategy
3.1.1 According to withdraw strategy, withdraw token
3.1.2 transfer token from handler
3.2.1 According to withdraw strategy withdraw token
3.2.2 transfer token from handler
4. update exchange rate & interest accrued
5. burn user's dToken

- ✓ Rebalance: attends to adjust the allocation of Tokens to different lending markets.



# Contracts and Functions:



Dispatcher
Public
getDepositStrategy(amount)
get deposit strategy, i.e. preset handlers and proportions.
getWithdrawStrategy(_token, amount)
get withdraw strategy, sort by liquidity.
getHandler()
get handlers connected.
admin
updateDefaultHandler(_defaultHandler)
update internal handler
updatePropotion( _handlers, _propotions)
update deposit proportions of handlers.
addHandler( _handlers)
add handler to dToken.
resetHandler(_handlers, _propotions)
reset handlers to new combination.
Proxy
dToken
Public
mint(_dst, _pie)
mint dToken.
burn(_src, _wad)
burn dToken by amount of dToken.
redeem(_src, _pie)
burn dToken by amount of underlying token.
admin
rebalance(address[] _withdraw, uint256[] _withdrawAmount, address[] _supply, uint256[] _supplyAmount)
rebalace token deposited in lending market.
setFeeRecipient(_newFeeRecipient)
set fee receiving account.
updateDispatcher(_newDispatcher)
update dispatcher connected.
updateOriginationFee(_sig, _newOriginationFee)
set origination fee of deposit / burn / redeem.
transferFee(_token, _amount)
transfer fee to fee receiver.
Public
getTokenBalance (_account)
get user token balance of underlying token.
getCurrentInterest(_account)
get interests earned by user.
getCurrentTotalInterest()
get total interests earned.
getHandler()
get handlers connected.
getTotalBalance()
get total balance of underlying token.
getLiquidity()
get total liquidity of lending market.
getExchangeRate()
get latest exchange rate of dToken.
Handler
Auth
deposit(_underlyingToken, _amount)
deposit underlying token to lending market.
withdraw(_underlyingToken, _amount)
withdraw underlying token from lending market.
admin
setdTokens(_newdTokens)
update dTokenAddress contract address.
disableToken(_underlyingToken)
disable underlying token in this handler.
enableToken(_underlyingToken)
enable underlying token in this handler.
approve(_underlyingToken)
approve underlying token to contract.
Public
getBalance(_underlyingToken)
get total balance of underlying token (by latest exchange rate)
getRealBalance(_underlyingToken)
get toal balance of underlying token (real time)
getLiquidity(_underlyingToken)
get underlying token liquidity in lending market(by latest exchange rate)
getRealLiquidity(_underlyingToken)
get underlying token liquidity in lending market(real time)
dTokenAddress
admin
setdTokensRelation(_tokens, _mappingdTokens)
map dToken and underlying token.
Public
getdToken(_token)
get dToken address by underlying token.