1. LenFi Smart Contract

# Lenfi - Permissionless Borrowing

## Introduction

We are building the product and its validators with a singular focus on creating a \*\*Permissionless system\*\*. This system is designed to be balanced, governed by smart contracts known as validators. It is an open system, free from dependencies on specific systems or individuals.

## Terminology

- \*\*NFT\*\*: A unique, non-fungible token whose uniqueness is guaranteed by a consumed UTXO reference.

- \*\*Pool\*\*: A UTXO with a Pool NFT, accompanied by a deposit and details (datum) locked in a `pool.ak` validator.

- \*\*Oracle\*\*: A UTXO with an Oracle NFT and details (datum) locked in a `collateral.ak` validator.

- \*\*Liquidity Provider (LP) Token\*\*: Fungible tokens that represent a share of the pool deposit.

- \*\*Collateral\*\*: Assets used as security to cover the value of loans taken from the pool.

- \*\*Interest Rate\*\*: Annual Percentage Rate (APR) charged on the loan amount.

- \*\*Loan\*\*: An agreement to borrow assets from a pool, governed by specific interest rates and collateral requirements.

- \*\*Pool Manager NFT\*\*: An NFT minted at the time of pool creation that allows for control over the stake address attached to `collateral.ak` and `pool.ak`.

## Off-chain validations

These are validation done off-chain, before pool appears on the client side. Failing to match any of below criteria would result in pool being dismissed as 'valid' and not displayed on the UI.

\*\*Pool validation\*\*

- Pool NFT is sent to `pool.ak` and Stake Credentials are correct (`pool\_stake.ak` script credentials)

- Pool NFT policyId matches `pool.ak`

- Minted LP tokens policyId matches `liquidity\_token.ak` policy. Also, it is recorded in pool datum lpToken polilcyId and asset name

- Pool config NFT is minted (policyId matches `pool\_config.ak`) and is sent to pool Config validator

- `pool\_stake.ak` script is locked to `leftovers.ak` for future contract references. With datum coressponding to Pool NFT.

\*\*Pool datum validation\*\*

- `collateralAddress`: validator ScriptCredential matches `collateral.ak`; Stake ScriptCredential matches pool stake credentials;

- `LoanCs`, `CollateralCs` are from approved list of borrowable and collateral assets

- `OracleCollateralNft/OracleLoanNft` is from approved list of oracle NFTs

- `lpToken` token name matches pool token name

- `poolNftName` matches pool NFT minted on the transaction

- `PoolConfigAssetname` matches pool NFT name

- `balance` match pool deposit amount and is used to calculate total\_lp\_tokens. Cannot be 0.

- `lent\_out` = 0

- `total\_lp\_tokens` match amount of LP tokens minted in the transaction

\*\*Oracle validation\*\*

- Oracle NFT is sent to oracle contract. Both match `oracle\_validator.ak` \*\*Oracle datum validation\*\*

- `poolNftCs` - hand checked valid DEX pool NFT associated to the token and ADA

- `oracleNftCs` - minted NFT CS

- `tokenACs` - matches DEX relation

- `tokenBCs` - is ADA ("","")

- `tokenAAmount` - current or approximate tokenA amount in a DEX

- `tokenBAmount` - current or approximate ADA amount in a DEX

- `expirationTime` - epoch time no later than 20 minutes from now

- `maturityTime` - epoch time no later than 20 minutes from now

\*\*Pool Config validation\*\*

- Pool Config NFT is sent to `pool\_config.ak` \*\*Config datum validation\*\* \_\*\*All the values are vetted by Lenfi team.\*\*\_

- `liquidation\_threshold` - is positive and ranging between 1,000,000 - 3,000,000. Cannot be changed

- `initial\_collateral\_ratio` - Is higher than `liquidation\_threshold`. Cannot be changed

- `pool\_fee` - is not negative.

- `loan\_fee\_details.tier\_1\_fee` - is not negative

- `loan\_fee\_details.tier\_1\_threshold` - is not negative

- `loan\_fee\_details.tier\_2\_fee` - higher than `tier\_1\_fee`

- `loan\_fee\_details.tier\_2\_threshold` - higher than `tier\_1\_threshold`

- `loan\_fee\_details.tier\_3\_fee` - higher than `tier\_2\_fee`

- `loan\_fee\_details.tier\_3\_threshold` - higher than `tier\_2\_threshold`

- `liquidation\_fee` - Is not negative

- `platform\_fee\_collector\_address`- is valid address

- `min\_transition` - is not negative

- `min\_loan` - is not negative

\*\*Reference Token validation\*\*

Reference token is minted at a time of protocol creation. Stores script hashes in a lock-forever script, replaces some script params and avoids circular imports thruought the protocol usage.

Offchain validates that Reference Token is and NFT locked in 'always fail' script with correct Datum values. `delegator\_nft\_policy` - Correct pool manager NFT policyId (`placeholder\_nft.ak` with param of `3`) `liquidations\_pkh` - Correct liquidations script hash (`leftovers.ak`) `pool\_script\_hash` - Correct pool script hash (`pool.ak`) `pool\_config\_script\_hash`- Correct pool config script hash (`pool\_config.ak`) `merge\_script\_hash`- Correct delayed merge script hash (`delayed\_merge.ak`)

## Governance control

Unique governance NFT is minted that has an access to control every pool config (stored in poolConfigValidator). NFT will be held by developer team before it is moved to DAO validator (see `pool\_config.ak` for more details). Governance NFT holder will have a right to change any `Pool Config Datum` except `liquidation\_threshold` and `initial\_collateral\_ratio`.

## Validators Definitions

- `collateral.ak`: Ensures fair loan issuance and repayment from the pool.

- `pool.ak`: Manages the proper utilization of user funds.

- `liquidity\_token.ak`: Oversees fair deposits and withdrawals from the pool.

- `pool\_config.ak`: Manages pool-specific parameters which can be adjusted based on off-chain consensus.

- `pool\_stake.ak`: Ensures that delegated ADA is withdrawn to the pool. Allows to control stake address.

- `order\_contract.ak`: Executes user orders fairly when direct execution with the pool is not possible.

- `oracle\_validator.ak`: Streams token prices to the protocol.

- `leftovers.ak`: Manages the return of any remaining collateral after loan repayment or liquidation.

- `delayed\_repayment\_merge.ak`: Allows to repay/liquidate the loan by sending funds to intermediate contract. This is handy when pool is busy.

- `placeholder\_nft.ak`: Ensures that minted NFTs are unique.

## Protocol Requirements

- Each pool only accepts one collateral asset defined in the pool datum

- LP token holders are entitled to a proportional share of the pool balance.

- An LP token's value in relation to the pool balance can never decrease; it can only appreciate due to accrued interest.

- Borrower NFT guarantees the right to reclaim 100% of collateral if the loan is repaid.

- Borrower NFT also guarantees the right to any remaining collateral, after loan and fees, if liquidation occurs.

- Loans must be overcollateralized.

- Loans can only be liquidated if undercollateralized at the time of liquidation.

- Liquidators are allowed to claim a percentage of the collateral as a fee during liquidation.

- Liquidateion fee is set in `pool\_config.ak` as a % of total collateral

- All loans must eventually be repaid; failure to do so will result in the loan becoming undercollateralized due to increasing interest.

- Delegation rewards can only be withdrawn to the pool.

- Pool Manager NFT allows delegation to any Stake Pool (SPO).

- Pool configuration (`pool\_config.ak`) values can be adjusted.

- Oracles are used to determine asset values during borrow, liquidate, and withdraw actions.

- Oracle data must be updated based on values from a Decentralized Exchange (DEX) and is considered expired if older than 30 minutes.

- Oracle data cannot be updated more frequently than every 10 minutes.

- A pool fee, set in `pool\_config`, is payable to the pool.

- Utilization rate (`lent\_out / (balance + lent\_out)`) determines the platform fee payable to the `platform\_fee\_collector\_address`.

- The pool datum must always accurately reflect the total supply of LP tokens n=n.

- Each pool must have unique: Pool NFT, pool address, stake address, and collateral address.

- The same stake address must be enforced for both the pool and the collateral.

- A pool can be destroyed if all balance is withdrawn and the Pool Manager NFT is burned.

- When user creates a batch order, they strictly define value they want to receive.

- Anyone can execute a batch order created by user.

## Use Cases

- Create a pool

- Delegate to an SPO

- Destroy a pool

- Deposit to a pool (may use `order\_contract.ak`)

- Withdraw from a pool (may use `order\_contract.ak`)

- Borrow (may use `order\_contract.ak`)

- Repay (may use `order\_contract.ak`)

- Liquidate (may use `order\_contract.ak`)

- Create an oracle

- Update oracle prices

## User Journeys

### Lender

1. Creates a pool.

2. Deposits funds into the pool and mints LP tokens.

3. Withdraws 50% from the pool and burns the corresponding LP tokens.

4. Withdraws the remaining 50% and destroys the pool.

### Borrower

1. Borrows from the pool, locks collateral, and mints a Borrower NFT.

2. Repays the loan, unlocks the collateral, and burns the Borrower NFT.

### Liquidator

1. Liquidates an undercollateralized loan by repaying the loan on behalf of the borrower.

2. Locks the remaining collateral for the borrower to claim later.

## Actions

### Creating a Pool

- A unique Pool NFT must be minted and locked in `pool.ak`.

- A unique Pool Manager NFT and Pool Config NFT must be minted.

- An initial deposit must be made to the pool.

- LP tokens must be minted to represent the initial deposit.

- LP token asset names must match the asset name of the Pool NFT.

- The `balance` and `total\_lp\_tokens` fields in the pool datum must be correctly initialized.

### Depositing to the Pool

- LP tokens must be minted to represent the deposit.

- The asset name of the LP tokens must match the asset name of the Pool NFT into which the deposit is made.

- The `balance` field in the pool datum must be incremented by the deposit amount.

- The `total\_lp\_tokens` field in the pool datum must be incremented by the amount of minted LP tokens.

### Withdrawing from the Pool

- The corresponding amount of LP tokens must be burned to represent the withdrawal.

- The asset name of the LP tokens must match the asset name of the Pool NFT from which the withdrawal is made.

- The asset name of the LP tokens must match the asset name of the Pool NFT into which the deposit is made.

- The `balance` field in the pool datum must be decremented by the withdrawal amount.

- The `total\_lp\_tokens` field in the pool datum must be decremented by the amount of burned LP tokens.

### Borrowing

- Mint a Borrower NFT upon initiating a loan.

- The loan amount should be smaller than the value of the collateral plus a buffer, defined as the liquidation threshold.

- Determine the interest rate as an Annual Percentage Rate (APR) for the entire loan duration.

- Securely lock collateral, which can either be claimed by the borrower upon repayment or liquidated if the loan becomes undercollateralized.

- Subject borrowers to a Loan-to-Value (LTV) ratio, as stipulated in the pool's `pool\_config.ak`.

- Specify the loan duration at the time of borrowing.

- Invoke `oracle\_validator.ak` to determine the current value of both the collateral and the borrowed asset.

- Use `leftovers.ak` to ensure that any remaining collateral is returned to the borrower once the loan is repaid or liquidated.

- Decrease the `balance` and increment `lent\_out` fields in the pool's datum.

### Repayment

- Only the holder of the Borrower NFT is authorized to repay the loan.

- Calculate the interest based on the duration the loan was active.

- Both the loan amount and interest should be repaid to the pool.

- Update the pool datum: increment the `balance` and decrement the `lent\_out` fields.

- Grant the borrower rights to reclaim 100% of the initially locked collateral.

### Liquidation

- Allow liquidation only when a loan is undercollateralized.

- Repay the loan and interest to the pool.

- Entitle the liquidator to a percentage of the collateral, where the percentage is specified in `pool\_config` (denominated by 1,000,000, e.g., 2% = 20,000).

- Make the remaining collateral claimable by the holder of the Borrower NFT.

### Delegation and Withdrawal

- Enable the Pool Manager NFT holder to delegate `pool.ak` and `collateral.ak` stake addresses to any Stake Pool Operator (SPO).

- Allow delegation rewards to be withdrawn only to the pool.

- Convert non-ADA assets to ADA at a fair market price (utilize the oracle) upon withdrawal.

### Oracle UTXO Update

- Update the Oracle UTXO strictly based on values from a Decentralized Exchange (DEX) UTXO.

- Restrict updates to a minimum interval of 10 minutes.

- Updated value is an average of datum from unlocked oracle UTXO and DEX

# Lenfi - Permissionless Borrowing

## Introduction

We are building the product and its validators with a singular focus on creating a \*\*Permissionless system\*\*. This system is designed to be balanced, governed by smart contracts known as validators. It is an open system, free from dependencies on specific systems or individuals.

## Terminology

- \*\*NFT\*\*: A unique, non-fungible token whose uniqueness is guaranteed by a consumed UTXO reference.

- \*\*Pool\*\*: A UTXO with a Pool NFT, accompanied by a deposit and details (datum) locked in a `pool.ak` validator.

- \*\*Oracle\*\*: A UTXO with an Oracle NFT and details (datum) locked in a `collateral.ak` validator.

- \*\*Liquidity Provider (LP) Token\*\*: Fungible tokens that represent a share of the pool deposit.

- \*\*Collateral\*\*: Assets used as security to cover the value of loans taken from the pool.

- \*\*Interest Rate\*\*: Annual Percentage Rate (APR) charged on the loan amount.

- \*\*Loan\*\*: An agreement to borrow assets from a pool, governed by specific interest rates and collateral requirements.

- \*\*Pool Manager NFT\*\*: An NFT minted at the time of pool creation that allows for control over the stake address attached to `collateral.ak` and `pool.ak`.

## Off-chain validations

These are validation done off-chain, before pool appears on the client side. Failing to match any of below criteria would result in pool being dismissed as 'valid' and not displayed on the UI.

\*\*Pool validation\*\*

- Pool NFT is sent to `pool.ak` and Stake Credentials are correct (`pool\_stake.ak` script credentials)

- Pool NFT policyId matches `pool.ak`

- Minted LP tokens policyId matches `liquidity\_token.ak` policy. Also, it is recorded in pool datum lpToken polilcyId and asset name

- Pool config NFT is minted (policyId matches `pool\_config.ak`) and is sent to pool Config validator

- `pool\_stake.ak` script is locked to `leftovers.ak` for future contract references. With datum coressponding to Pool NFT.

\*\*Pool datum validation\*\*

- `collateralAddress`: validator ScriptCredential matches `collateral.ak`; Stake ScriptCredential matches pool stake credentials;

- `LoanCs`, `CollateralCs` are from approved list of borrowable and collateral assets

- `OracleCollateralNft/OracleLoanNft` is from approved list of oracle NFTs

- `lpToken` token name matches pool token name

- `poolNftName` matches pool NFT minted on the transaction

- `PoolConfigAssetname` matches pool NFT name

- `balance` match pool deposit amount and is used to calculate total\_lp\_tokens. Cannot be 0.

- `lent\_out` = 0

- `total\_lp\_tokens` match amount of LP tokens minted in the transaction

\*\*Oracle validation\*\*

- Oracle NFT is sent to oracle contract. Both match `oracle\_validator.ak` \*\*Oracle datum validation\*\*

- `poolNftCs` - hand checked valid DEX pool NFT associated to the token and ADA

- `oracleNftCs` - minted NFT CS

- `tokenACs` - matches DEX relation

- `tokenBCs` - is ADA ("","")

- `tokenAAmount` - current or approximate tokenA amount in a DEX

- `tokenBAmount` - current or approximate ADA amount in a DEX

- `expirationTime` - epoch time no later than 20 minutes from now

- `maturityTime` - epoch time no later than 20 minutes from now

\*\*Pool Config validation\*\*

- Pool Config NFT is sent to `pool\_config.ak` \*\*Config datum validation\*\* \_\*\*All the values are vetted by Lenfi team.\*\*\_

- `liquidation\_threshold` - is positive and ranging between 1,000,000 - 3,000,000. Cannot be changed

- `initial\_collateral\_ratio` - Is higher than `liquidation\_threshold`. Cannot be changed

- `pool\_fee` - is not negative.

- `loan\_fee\_details.tier\_1\_fee` - is not negative

- `loan\_fee\_details.tier\_1\_threshold` - is not negative

- `loan\_fee\_details.tier\_2\_fee` - higher than `tier\_1\_fee`

- `loan\_fee\_details.tier\_2\_threshold` - higher than `tier\_1\_threshold`

- `loan\_fee\_details.tier\_3\_fee` - higher than `tier\_2\_fee`

- `loan\_fee\_details.tier\_3\_threshold` - higher than `tier\_2\_threshold`

- `liquidation\_fee` - Is not negative

- `platform\_fee\_collector\_address`- is valid address

- `min\_transition` - is not negative

- `min\_loan` - is not negative

\*\*Reference Token validation\*\*

Reference token is minted at a time of protocol creation. Stores script hashes in a lock-forever script, replaces some script params and avoids circular imports thruought the protocol usage.

Offchain validates that Reference Token is and NFT locked in 'always fail' script with correct Datum values. `delegator\_nft\_policy` - Correct pool manager NFT policyId (`placeholder\_nft.ak` with param of `3`) `liquidations\_pkh` - Correct liquidations script hash (`leftovers.ak`) `pool\_script\_hash` - Correct pool script hash (`pool.ak`) `pool\_config\_script\_hash`- Correct pool config script hash (`pool\_config.ak`) `merge\_script\_hash`- Correct delayed merge script hash (`delayed\_merge.ak`)

## Governance control

Unique governance NFT is minted that has an access to control every pool config (stored in poolConfigValidator). NFT will be held by developer team before it is moved to DAO validator (see `pool\_config.ak` for more details). Governance NFT holder will have a right to change any `Pool Config Datum` except `liquidation\_threshold` and `initial\_collateral\_ratio`.

## Validators Definitions

- `collateral.ak`: Ensures fair loan issuance and repayment from the pool.

- `pool.ak`: Manages the proper utilization of user funds.

- `liquidity\_token.ak`: Oversees fair deposits and withdrawals from the pool.

- `pool\_config.ak`: Manages pool-specific parameters which can be adjusted based on off-chain consensus.

- `pool\_stake.ak`: Ensures that delegated ADA is withdrawn to the pool. Allows to control stake address.

- `order\_contract.ak`: Executes user orders fairly when direct execution with the pool is not possible.

- `oracle\_validator.ak`: Streams token prices to the protocol.

- `leftovers.ak`: Manages the return of any remaining collateral after loan repayment or liquidation.

- `delayed\_repayment\_merge.ak`: Allows to repay/liquidate the loan by sending funds to intermediate contract. This is handy when pool is busy.

- `placeholder\_nft.ak`: Ensures that minted NFTs are unique.

## Protocol Requirements

- Each pool only accepts one collateral asset defined in the pool datum

- LP token holders are entitled to a proportional share of the pool balance.

- An LP token's value in relation to the pool balance can never decrease; it can only appreciate due to accrued interest.

- Borrower NFT guarantees the right to reclaim 100% of collateral if the loan is repaid.

- Borrower NFT also guarantees the right to any remaining collateral, after loan and fees, if liquidation occurs.

- Loans must be overcollateralized.

- Loans can only be liquidated if undercollateralized at the time of liquidation.

- Liquidators are allowed to claim a percentage of the collateral as a fee during liquidation.

- Liquidateion fee is set in `pool\_config.ak` as a % of total collateral

- All loans must eventually be repaid; failure to do so will result in the loan becoming undercollateralized due to increasing interest.

- Delegation rewards can only be withdrawn to the pool.

- Pool Manager NFT allows delegation to any Stake Pool (SPO).

- Pool configuration (`pool\_config.ak`) values can be adjusted.

- Oracles are used to determine asset values during borrow, liquidate, and withdraw actions.

- Oracle data must be updated based on values from a Decentralized Exchange (DEX) and is considered expired if older than 30 minutes.

- Oracle data cannot be updated more frequently than every 10 minutes.

- A pool fee, set in `pool\_config`, is payable to the pool.

- Utilization rate (`lent\_out / (balance + lent\_out)`) determines the platform fee payable to the `platform\_fee\_collector\_address`.

- The pool datum must always accurately reflect the total supply of LP tokens n=n.

- Each pool must have unique: Pool NFT, pool address, stake address, and collateral address.

- The same stake address must be enforced for both the pool and the collateral.

- A pool can be destroyed if all balance is withdrawn and the Pool Manager NFT is burned.

- When user creates a batch order, they strictly define value they want to receive.

- Anyone can execute a batch order created by user.

## Use Cases

- Create a pool

- Delegate to an SPO

- Destroy a pool

- Deposit to a pool (may use `order\_contract.ak`)

- Withdraw from a pool (may use `order\_contract.ak`)

- Borrow (may use `order\_contract.ak`)

- Repay (may use `order\_contract.ak`)

- Liquidate (may use `order\_contract.ak`)

- Create an oracle

- Update oracle prices

## User Journeys

### Lender

1. Creates a pool.

2. Deposits funds into the pool and mints LP tokens.

3. Withdraws 50% from the pool and burns the corresponding LP tokens.

4. Withdraws the remaining 50% and destroys the pool.

### Borrower

1. Borrows from the pool, locks collateral, and mints a Borrower NFT.

2. Repays the loan, unlocks the collateral, and burns the Borrower NFT.

### Liquidator

1. Liquidates an undercollateralized loan by repaying the loan on behalf of the borrower.

2. Locks the remaining collateral for the borrower to claim later.

## Actions

### Creating a Pool

- A unique Pool NFT must be minted and locked in `pool.ak`.

- A unique Pool Manager NFT and Pool Config NFT must be minted.

- An initial deposit must be made to the pool.

- LP tokens must be minted to represent the initial deposit.

- LP token asset names must match the asset name of the Pool NFT.

- The `balance` and `total\_lp\_tokens` fields in the pool datum must be correctly initialized.

### Depositing to the Pool

- LP tokens must be minted to represent the deposit.

- The asset name of the LP tokens must match the asset name of the Pool NFT into which the deposit is made.

- The `balance` field in the pool datum must be incremented by the deposit amount.

- The `total\_lp\_tokens` field in the pool datum must be incremented by the amount of minted LP tokens.

### Withdrawing from the Pool

- The corresponding amount of LP tokens must be burned to represent the withdrawal.

- The asset name of the LP tokens must match the asset name of the Pool NFT from which the withdrawal is made.

- The asset name of the LP tokens must match the asset name of the Pool NFT into which the deposit is made.

- The `balance` field in the pool datum must be decremented by the withdrawal amount.

- The `total\_lp\_tokens` field in the pool datum must be decremented by the amount of burned LP tokens.

### Borrowing

- Mint a Borrower NFT upon initiating a loan.

- The loan amount should be smaller than the value of the collateral plus a buffer, defined as the liquidation threshold.

- Determine the interest rate as an Annual Percentage Rate (APR) for the entire loan duration.

- Securely lock collateral, which can either be claimed by the borrower upon repayment or liquidated if the loan becomes undercollateralized.

- Subject borrowers to a Loan-to-Value (LTV) ratio, as stipulated in the pool's `pool\_config.ak`.

- Specify the loan duration at the time of borrowing.

- Invoke `oracle\_validator.ak` to determine the current value of both the collateral and the borrowed asset.

- Use `leftovers.ak` to ensure that any remaining collateral is returned to the borrower once the loan is repaid or liquidated.

- Decrease the `balance` and increment `lent\_out` fields in the pool's datum.

### Repayment

- Only the holder of the Borrower NFT is authorized to repay the loan.

- Calculate the interest based on the duration the loan was active.

- Both the loan amount and interest should be repaid to the pool.

- Update the pool datum: increment the `balance` and decrement the `lent\_out` fields.

- Grant the borrower rights to reclaim 100% of the initially locked collateral.

### Liquidation

- Allow liquidation only when a loan is undercollateralized.

- Repay the loan and interest to the pool.

- Entitle the liquidator to a percentage of the collateral, where the percentage is specified in `pool\_config` (denominated by 1,000,000, e.g., 2% = 20,000).

- Make the remaining collateral claimable by the holder of the Borrower NFT.

### Delegation and Withdrawal

- Enable the Pool Manager NFT holder to delegate `pool.ak` and `collateral.ak` stake addresses to any Stake Pool Operator (SPO).

- Allow delegation rewards to be withdrawn only to the pool.

- Convert non-ADA assets to ADA at a fair market price (utilize the oracle) upon withdrawal.

### Oracle UTXO Update

- Update the Oracle UTXO strictly based on values from a Decentralized Exchange (DEX) UTXO.

- Restrict updates to a minimum interval of 10 minutes.

- Updated value is an average of datum from unlocked oracle UTXO and DEX

