# Ref SDK

This SDK is designed to assist developers when interacting with the main functions of the protocol. Main functions that can be defined as:

- Trade: Swap tokens with our Automated Market Maker (AMM)
- Pool: Add/Remove liquidity and earn revenue from swap fee (Coming soon)
- Farm: Stale LP tokens into farms and earn liquidity incentives (Coming soon)
- Boost Farm: Stake LOVE tokens to get boosted liquidity incentives (Coming soon)
- Stake: Stake REF tokens to earn fees generated by the protocol (Coming soon)
- Vote: Lock REF<>NEAR LP tokens to get veTokens and participate in the governance of the protocol and the allocation of liquidity incentives (Coming soon)

## Install

yarn: `yarn add @ref-finance/ref-sdk`

npm: `npm install @ref-finance/ref-sdk`

## Initialization

Ref SDK identifies env variable NEAR_ENV or REACT_APP_REF_SDK_ENV to get global configuration.

```plain
export function getConfig(
env: string | undefined = process.env.NEAR_ENV ||
process.env.REACT_APP_REF_SDK_ENV
) {
switch (env) {
case 'mainnet':
return {
networkId: 'mainnet',
nodeUrl: 'https://rpc.mainnet.near.org',
walletUrl: 'https://wallet.near.org',
WRAP_NEAR_CONTRACT_ID: 'wrap.near',
REF_FI_CONTRACT_ID: 'v2.ref-finance.near',
REF_TOKEN_ID: 'token.v2.ref-finance.near',
indexerUrl: 'https://indexer.ref.finance',
explorerUrl: 'https://testnet.nearblocks.io',
REF_DCL_SWAP_CONTRACT_ID: '',
};
case 'testnet':
return {
networkId: 'testnet',
nodeUrl: 'https://rpc.testnet.near.org',
walletUrl: 'https://wallet.testnet.near.org',
indexerUrl: 'https://testnet-indexer.ref-finance.com',
WRAP_NEAR_CONTRACT_ID: 'wrap.testnet',
REF_FI_CONTRACT_ID: 'ref-finance-101.testnet',
REF_TOKEN_ID: 'ref.fakes.testnet',
explorerUrl: 'https://testnet.nearblocks.io',
REF_DCL_SWAP_CONTRACT_ID: 'dcl.ref-dev.testnet',
};
default:
return {
networkId: 'mainnet',
nodeUrl: 'https://rpc.mainnet.near.org',
walletUrl: 'https://wallet.near.org',
REF_FI_CONTRACT_ID: 'v2.ref-finance.near',
WRAP_NEAR_CONTRACT_ID: 'wrap.near',
REF_TOKEN_ID: 'token.v2.ref-finance.near',
indexerUrl: 'https://indexer.ref.finance',
explorerUrl: 'https://nearblocks.io',
REF_DCL_SWAP_CONTRACT_ID: '',
};
}
}
```

Also, the SDK provides `init_env` to switch application environment. We'd better call it at the entrance of the application, then the entire application environment switch takes effect. If you're not sure where to call, you can call in more than one place.

Example

```
init_env('testnet');
init_env('mainnet');
```

## Ref V1 Swap

### Tokens

#### ftGetTokenMetadata

Get token metadata.

Parameters

```plain
id: string;
```

Example

```plain
const WrapNear = await ftGetTokenMetadata('wrap.testnet');
```

Response

```plain
{
decimals: 24;
icon: null;
id: 'wrap.testnet';
name: 'Wrapped NEAR fungible token';
reference: null;
reference_hash: null;
spec: 'ft-1.0.0';
symbol: 'wNEAR';
}
```

#### ftGetTokensMetadata

Get tokens metadata and set token id as index.

Parameters

```plain
tokenIds: string[]
```

Example

```plain
const tokensMetadata = await ftGetTokensMetadata([
'ref.fakes.testnet',
'wrap.testnet',
]);
```

Response

```plain
{
"ref.fakes.testnet":{
decimals: 18
icon: "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' viewBox='16 24 248 248' style='background: %23000'%3E%3Cpath d='M164,164v52h52Zm-45-45,20.4,20.4,20.6-20.6V81H119Zm0,18.39V216h41V137.19l-20.6,20.6ZM166.5,81H164v33.81l26.16-26.17A40.29,40.29,0,0,0,166.5,81ZM72,153.19V216h43V133.4l-11.6-11.61Zm0-18.38,31.4-31.4L115,115V81H72ZM207,121.5h0a40.29,40.29,0,0,0-7.64-23.66L164,133.19V162h2.5A40.5,40.5,0,0,0,207,121.5Z' fill='%23fff'/%3E%3Cpath d='M189 72l27 27V72h-27z' fill='%2300c08b'/%3E%3C/svg%3E%0A"
id: "ref.fakes.testnet"
name: "Ref Finance Token"
reference: null
reference_hash: null
spec: "ft-1.0.0"
symbol: "REF"
},
"wrap.testnet":{
decimals: 24
icon: null
id: "wrap.testnet"
name: "Wrapped NEAR fungible token"
reference: null
reference_hash: null
spec: "ft-1.0.0"
symbol: "wNEAR"
}
}
```

### Pools

#### fetchAllPools

Fetch all existing pools, including vanilla/simple pools, stable pools and rated pools (designed for yield-bearing tokens).

Parameters

perPage?： number

Example

```plain
const { ratedPools, unRatedPools, simplePools } = await fetchAllPools();
const { ratedPools, unRatedPools, simplePools } = await fetchAllPools(200);
```

Response

```plain
{
ratedPools:[{
fee: 5,
id: 568,
pool_kind: "RATED_SWAP",
shareSupply: "80676034815429711745720012070",
supplies:{
"meta-v2.pool.testnet": "1298314415249170366960739764"
"wrap.testnet": "80182803630538035347294614770"
},
token0_ref_price: undefined,
tokenIds: ["meta-v2.pool.testnet", "wrap.testnet"],
tvl: undefined
},...]
unRatedPools:[...],
simplePools:[...],
}
```

#### getStablePools

We define `unRatedPools` and `ratedPools` as `stablePool`. You can use this function to get details of stable pools.

Parameters

```plain
stablePools: Pool[]
```

Example

```plain
const stablePools: Pool[] = unRatedPools.concat(ratedPools);

const stablePoolsDetail: StablePool[] = await getStablePools(stablePools);
```

Response

```plain
[
{
amounts: ['1298314415249170366960739764',
'8018280363053803534729461 4770'],
amp: 240,
c_amounts:
["1298314415249170366960739764","80182803630538035347294614770"],
decimals:[24,24],
id: 568,
pool_kind: "RATED_SWAP",
rates:["1972101024157559347385372","1000000000000000000000000"],
shares_total_supply: "80676034815429711745720012070",
token_account_ids:["meta-v2.pool.testnet","wrap.testnet"],
total_fee:5
},
...
]
```

### Swap

#### estimateSwap

Get token output amount and corresponding route.

As there is a memory limitation on Ledger, note that we set `enableSmartRouting` option for developers.

This function integrates a smart routing algorithm, designed to deliver the best output token amount.

Parameters

```plain
interface SwapParams {
tokenIn: TokenMetadata;
tokenOut: TokenMetadata;
amountIn: string;
simplePools: Pool[];
options?: SwapOptions;
}

interface SwapOptions {
enableSmartRouting?: boolean;
stablePools?: Pool[];
stablePoolsDetail?: StablePool[];
}
```

Example (enableSmartRouting == false)

```plain
// enableSmartRouting as FALSE, swap from Ref to wNear, with amount 1
const tokenIn = await ftGetTokenMetadata('ref.fakes.testnet');
const tokenOut = await ftGetTokenMetadata('wrap.testnet');

const swapTodos: EstimateSwapView[] = await estimateSwap({
tokenIn,
tokenOut,
amountIn: '1',
simplePools,
});
```

Response (enableSmartRouting == false)

```plain
// enableSmartRouting as FALSE, swap from Ref to wNear, with amount 1

[
{
estimate: '0.7338604246699393',
inputToken: 'ref.fakes.testnet',
outputToken: 'wrap.testnet',
pool: {
fee: 30,
id: 38,
partialAmountIn: '1000000000000000000',
pool_kind: 'SIMPLE_POOL',
shareSupply: '1000587315520795219676332',
supplies: {
'ref.fakes.testnet': '7789776060978885018',
'wrap.testnet': '6467670222256390319335181',
},
token0_ref_price: undefined,
tokenIds: (2)[('ref.fakes.testnet', 'wrap.testnet')],
tvl: undefined,
},
},
];
```

Example (enableSmartRouting == true)

```plain
// enableSmartRouting as TRUE, swap from Ref to wNear, with amount 1
const tokenIn = await ftGetTokenMetadata('ref.fakes.testnet');
const tokenOut = await ftGetTokenMetadata('wrap.testnet');

const options: SwapOptions = {
enableSmartRouting: true,
stablePools,
stablePoolsDetail,
};

const swapTodos: EstimateSwapView[] = await estimateSwap({
tokenIn,
tokenOut,
amountIn: '1',
simplePools,
options,
});
```

Response (enableSmartRouting == true)

```plain
// enableSmartRouting as true, swap from Ref to wNear, with amount 1
```

```
[
{
estimate: "0.00022532154427509590237135556972009167",
inputToken: "ref.fakes.testnet",
outputToken: "nusdt.ft-fin.testnet",
pool:{
id: 341,
partialAmountIn: "836859596261755688",
tokenIds: ["ref.fakes.testnet","nusdt.ft-fin.testnet"],
...
},
...
},{
estimate: "0.5203255327171591155634413370660973429264",
inputToken: "nusdt.ft-fin.testnet",
outputToken: "wrap.testnet",
pool:{
id: 1625,
tokenIds: ["nusdt.ft-fin.testnet","wrap.testnet"],
...
},
...
},{
estimate: "5.3206339674140066489000963525772735539612",
inputToken: "ref.fakes.testnet",
outputToken: "usdn.testnet",
pool:{
id: 376,
tokenIds: ["usdn.testnet","ref.fakes.testnet"],
partialAmountIn: "163140403738244312",
...
},
...
},{
estimate: "0.2036473132202839680683206178037243543302",
inputToken: "usdn.testnet",
outputToken: "wrap.testnet",
pool:{
id: 385,
tokenIds: ["usdn.testnet","wrap.testnet"],
...
},
...
}
]
```

#### getExpectedOutputFromSwapTodos

Get token output amount from swapTodos.

Parameters

```plain
(swapTodos: EstimateSwapView[], outputToken: string)
```

Example

```plain
const swapTodos: EstimateSwapView[] = await estimateSwap({
tokenIn,
tokenOut,
amountIn: '1',
simplePools,
options,
});

const amountOut:string = getExpectedOutputFromSwapTodos(swapTodos,
tokenOut.id);
```

Response

```plain
"0.723972845937443"
```

#### getPoolEstimate

Get token output amount from one single pool. Method can be used for simple and stable pools.

Parameters

```plain
```

```
{
tokenIn: TokenMetadata;
tokenOut: TokenMetadata;
amountIn: string;
pool: Pool;
// please input stablePoolDetail if you want estimate output on stable pool or the pool will be recognized as simple pool
stablePoolDetail?: StablePool;
}
```

Example (on simple pool)

```plain
// estimate on simple Pool
const estimate = await getPoolEstimate({
tokenIn,
tokenOut,
amountIn: '1',
pool,
});
```

Response (on simple pool)

```plain
// estimate on simple pool, swap from Ref to wNear, with amount 1
{
estimate: "0.7338604246699393",
inputToken: "ref.fakes.testnet",
outputToken: "wrap.testnet",
pool:{
fee: 30,
id: 38,
partialAmountIn: "1000000000000000000",
pool_kind: "SIMPLE_POOL",
shareSupply: "1000587315520795219676332",
supplies: {"ref.fakes.testnet": '7789776060978885018', "wrap.testnet": '6467670222256390319335181'},
token0_ref_price: undefined,
tokenIds: (2) ['ref.fakes.testnet', 'wrap.testnet'],
tvl: undefined
}
}
```

Example (on stable pool)

```plain
// estimate on stable pool
const estimate = await getPoolEstimate({
tokenIn,
tokenOut,
amountIn: '1',
pool: stablePool,
stablePoolDetail: stablePoolDetail,
});
```

Response (on stable pool)

```plain
// estimate on stable pool, swap from stNear to wNear, with amount 1
{
estimate: "2.4898866773442284",
inputToken: "meta-v2.pool.testnet",
noFeeAmountOut: "2.491132243465961",
outputToken: "wrap.testnet",
pool:{
amounts:
["1298314415249170366960739764","8018280363053803534729461477 0"],
amp: 240,
c_amounts:
["1298314415249170366960739764","8018280363053803534729461477 0"],
decimals:[24,24],
id: 568,
pool_kind: "RATED_SWAP",
rates:["1972204647926836788049038","1000000000000000000000000"],
shares_total_supply: "80676034815429711745720012070",
token_account_ids:["meta-v2.pool.testnet", "wrap.testnet"],
total_fee: 5,
}
}
```

### Transactions

#### instantSwap

Set up transactions through swap routes. Please ensure that the AccountId has an active balance storage in the token-in contract, otherwise the transaction will fail and the user will lose the token input amount.

Parameters

```plain
{
tokenIn: TokenMetadata;
tokenOut: TokenMetadata;
amountIn: string;
slippageTolerance: number;
swapTodos: EstimateSwapView[];
AccountId: string;
}
```

Example

```plain
const transactionsRef: Transaction[] = await instantSwap({
tokenIn,
tokenOut,
amountIn: '1',
swapTodos,
slippageTolerance = 0.01,
AccountId: 'your-account-id.testnet'
});
```

Response

```plain
[
{
functionCalls: [
{
amount: '0.000000000000000000000001',
args: {
amount: '1000000000000000000',
msg:
'{"force":0,"actions":
[{"pool_id":38,"token_in":"ref.fakes.testnet","token_out":"wrap.testnet","amount_in":
"1000000000000000000","min_amount_out":"730191122546589600000000"}]}',
receiver_id: 'ref-finance-101.testnet',
},
gas: '180000000000000',
methodName: 'ft_transfer_call',
},
],
receiverId: 'ref.fakes.testnet',
},
];
```

#### getSignedTransactionsByMemoryKey (Node)

In the local env, developers can add credentials by `near login` .

This function utilizes credentials stored in the local env to sign transactions.

Parameters

```plain
{
transactionsRef: Transaction[];
AccountId: string;
keyPath: string;
}
```

Example

```plain
const signedTransactions:nearTransactions.SignedTransaction[] =
getSignedTransactionsByMemoryKey({
transactionsRef;
AccountId: "your-account-id.testnet",
keyPath: "/.near-credentials/testnet/your-account-id.testnet.json"
})
```

Response

```plain
[
SignedTransaction {
transaction: Transaction {
signerId: 'your-account-id.testnet',
publicKey: [PublicKey],
nonce: 91940092000042,
receiverId: 'ref.fakes.testnet',
actions: [Array],
blockHash: <Buffer 45 e5 fd 36 87 3b 10 59 81 d9 a7 b5 20 c7 29 33 f7 27 48 59
06 90 ca 8a 17 03 5c 25 f2 76 ab 7c>
},
signature: Signature { keyType: 0, data: [Uint8Array] }
}
]
```

#### sendTransactionsByMemoryKey (Node)

This function utilizes credentials stored in the local env to send transactions.

Parameters

```plain
{
signedTransactions: nearTransactions.SignedTransaction[];
}
```

Example

```plain
sendTransactionsByMemoryKey({
signedTransactions,
});
```

Response

```plain
[
{
receipts_outcome: [
[Object], [Object],
[Object], [Object],
[Object], [Object],
[Object], [Object],
[Object], [Object]
],
status: { SuccessValue: 'xxxxxx' },
transaction: {
actions: [Array],
hash: 'xxxxxxxx',
nonce: 91940092000042,
public_key: 'ed25519:xxxxxxxx',
receiver_id: 'ref.fakes.testnet',
signature: 'ed25519:xxxxxxxxxxxxx',
signer_id: 'your-account-id.testnet'
},
transaction_outcome: {
block_hash: '3QXEF941UvsHzXrMhwbchk7wmy3qmPNs3w9gkfvW84QK',
id: 'xxxxxxxx',
outcome: [Object],
proof: []
}
}
]
```

## Ref Swap Widget

### Description

The Ref Swap Widget is a useful tool, allowing any third party service to access Ref's liquidity. Users of ecosystem dapps have the ability to swap via the Widget, without the need to go to Ref app, thus improving the user experience.

Here are some use cases:

- Swapping stablecoins for your project's token
- Swapping tokens to lend, farm or stake
- Swapping one token for a specific token, which can be used to buy a NFT in the associated marketplace

Using the Ref Swap Widget, with a few customizations, developers can integrate the Swap funtion directly into their dapps. Both mobile and/or website version are available.
![图片](https://user-images.githubusercontent.com/50706666/199178215-f2b184dc-f683-4740-af9f-fd67efd41503.png)

For the default theme, developers can chose between the light mode and dark mode.

More themes can be selected: [Click here to check them on figma](https://www.figma.com/file/v069nTXfE8pXDJQcDcC5wl/Swap-Widget?node-id=0%3A1).

To integrate the Ref Swap Widget, please follow this guide.

### Getting started

A QuickStart of Ref Swap component.

#### Props

```plain
export interface SwapWidgetProps {
theme?: Theme;
extraTokenList?: string[];
onSwap: (transactionsRef: Transaction[]) => void;
onDisConnect: () => void;
width: string;
height?: string;
enableSmartRouting?: boolean;
className?: string;
darkMode?: boolean;
connection: {
AccountId: string;
isSignedIn: boolean;
};
defaultTokenIn?: string;
defaultTokenOut?: string;
transactionState?: {
state: 'success' | 'fail' | null;
tx?: string;
detail?: string;
};
onConnect: () => void;
}
```

- theme: widget theme for customization.
- extraTokenList: introduce extra tokens with ref whitelist into default token list in the widget.
- onSwap: Swap button triggers this function.
- width: width of widget component.
- height: height of widget component.
- enableSmartRouting: option to choose if enable smart routing in swap routes

estimation.
- className: extra className added to widget component.
- darkMode: if true, will automatically set theme to default dark mode.
- connection: connection to wallets, input { AccountId:"", isSignedIn:false } if wallet not connected.
- defaultTokenIn: default token-in.
- defaultTokenOut: default token-out.
- transactionState: entry to input transaction states after you send transactions.
- state: denote if last transaction is failed or successfull.
- setState: used to change setState to interact with pop-up.
- tx: will add link to near explorer according to this tx.
- detail: you could input some tips to show on sucess pop-up.

![111](https://user-images.githubusercontent.com/50706666/199178453-8d09be3f-5a00-4b62-a6f1-af42ce4beae6.png)

- onDisConnect: Disconnect button triggers this function.
- onConnect: Connect to Near Wallet button triggers this function.

### Usage

#### Theme

```plain
export interface Theme {
container: string; // container background
buttonBg: string; // button background
primary: string; // primary theme color
secondary: string; // secondary theme color
borderRadius: string; // border radius
fontFamily: string; // font family
hover: string; // hovering color
active: string; // active color
secondaryBg: string; // secondary background color
borderColor: string; // border color
iconDefault: string; // default icon color
iconHover: string; // icon hovering color
refIcon?: string; // ref icon color, default to be black
}

export const defaultTheme: Theme = {
container: '#FFFFFF',
buttonBg: '#00C6A2',
primary: '#000000',
secondary: '#7E8A93',
borderRadius: '4px',
fontFamily: 'sans-serif',
hover: 'rgba(126, 138, 147, 0.2)',
active: 'rgba(126, 138, 147, 0.2)',
secondaryBg: '#F7F7F7',
borderColor: 'rgba(126, 138, 147, 0.2)',
iconDefault: '#7E8A93',
iconHover: '#B7C9D6',
};

export const defaultDarkModeTheme: Theme = {
container: '#26343E',
buttonBg: '#00C6A2',
primary: '#FFFFFF',
secondary: '#7E8A93',
borderRadius: '4px',
fontFamily: 'sans-serif',
hover: 'rgba(126, 138, 147, 0.2)',
active: 'rgba(126, 138, 147, 0.2)',
secondaryBg: 'rgba(0, 0, 0, 0.2)',
borderColor: 'rgba(126, 138, 147, 0.2)',
iconDefault: '#7E8A93',
iconHover: '#B7C9D6',
refIcon: 'white',
};
```

#### Component

```plain
// an example of combining SwapWidget with wallet-selector
import * as React from 'react';
import { SwapWidget } from '@ref-finance/ref-sdk';

// please check on wallet-selector example about how to set WalletSelectorContext
import { useWalletSelector } from './WalletSelectorContext';

import { WalletSelectorTransactions, NotLoginError } from '@ref-finance/ref-sdk';

export const Widget = ()=>{

const { modal, selector, accountId } = useWalletSelector();

const [swapState, setSwapState] = React.useState<'success' | 'fail' | null>(
null
```

```
);
const [tx, setTx] = React.useState<string | undefined>(undefined);
React.useEffect(() => {
const errorCode = new URLSearchParams(window.location.search).get(
'errorCode'
);

const transactions = new URLSearchParams(window.location.search).get(
'transactionHashes'
);

const lastTX = transactions?.split(',').pop();

setTx(lastTX);

setSwapState(!!errorCode ? 'fail' : !!lastTX ? 'success' : null);

window.history.replaceState(
{},
'',
window.location.origin + window.location.pathname
);
}, []);

const onSwap = async (transactionsRef: Transaction[]) => {
const wallet = await selector.wallet();
if (!accountId) throw NotLoginError;

wallet.signAndSendTransactions(
WalletSelectorTransactions(transactionsRef, accountId)
);
};

const onConnect = () => {
modal.show();
};

const onDisConnect = async () => {
const wallet = await selector.wallet();
return await wallet.signOut();
};

return (
<SwapWidget
onSwap={onSwap}
```

```
onDisConnect={onDisConnect}
width={'400px'}
connection={{
AccountId: accountId || '',
isSignedIn: !!accountId,
}}
transactionState={{
state: swapState,
setState: setSwapState,
tx,
detail: '(success details show here)',
}}
enableSmartRouting={true}
onConnect={onConnect}
defaultTokenIn={'wrap.testnet'}
defaultTokenOut={'ref.fakes.testnet'}
/>
);
}
```

### Integrating Ref Swap function using the SDK

The SDK provides more flexibility/options.

The default Swap version can serve as a reference. [Click here to check on figma.](https://www.figma.com/file/v069nTXfE8pXDJQcDcC5wl/Swap-Widget?node-id=0%3A1)

For more details about the SDK, please refer to [here].

SDK integration tips:

- You can use 'ftGetTokensMetadata' function to get all available tokens, and list them in the token selector to allow users to select their trading pair. If you do not need all available tokens, you can limit the list at the frontend level, thus only displaying specific tokens such as REF, NEAR, and USDT, for example.
- Please pay attention to the user's 'tokenIn' balance. If the balance is zero, you should **DISABLE** the swap button.
- You can let the user set the slippage, or you can pre set a default number.
- For a better user experience, before the execution of the swap, you can show more details about the swap (ex: fee, rate, route, etc.), allowing users to take better data-driven decisions.
- You can redirect the user to the NEAR Explorer, once the transaction is confirmed.

## Ref V2(DCL) Swap

#### An overview of Ref V2

The launch of concentrated liquidity AMM is an achievement for Ref Finance. In collaboration with Izumi Finance and Arctic, Ref is glad to introduce discretized concentrated liquidity and limit order two new key features to the NEAR ecosystem. Using REF SDK, developers can dig more opportunities and implement various trading strategies.Before introducing SDK details, let's have an overview of V2 exciting features.

#### Discretized concentrated liquidity

Discretized concentrated liquidity can improve as much as 4000x higher capital efficiency for liquidity providers(LPs).

When adding liquidity, LPs are can allocate their capital to a certain price range thus providing greater amounts of liquidity at this range. The fees earned are decided by the volumes swapped in the price range. Logically, if you want to earn more, you should set the price range most desired by the market.

Here are the advantages of using Ref V2 concentrated liquidity for LPS.

- **Efficient swap fee earnings and less impermanent loss** by setting a price range
- **Improved fee structure:** only LPs (and the protocol itself) can benefit from swap fees, <u>which means an LP can also benefit from a transaction between a taker and market makers as long the transaction closes at the price within the range set by him/her. This is quite different from Uniswap V3 by giving more benefits to LPS.</u>
- **Versatile strategies for different pairs:** according to correlations of trading pairs, market demand and fee tiers, versatile strategies can be designed to make a profit and hedge against risk.

#### Enhanced limit order

Izumi's algorithm of constant sum formula on small price range fragment enables an enhanced limit order function that is different than Uniswap V3. The following remarkable improvements are noticeable.

- **No price restrictions when placing a limit order:** Users can place limit buy orders at a price higher than current price or sell orders lower than current price. Ref V2 would auto match with best price first, similar to CLOBs (central limit order books) seen on centralized exchanges.
- **No need to keep an eye on order progress:** As a kind of one-way liquidity, users can claim their order earning at any time without any concern that the earned token would be reverted.

- **CEX order-book style user experience.**

Enhanced limit order will give more convenience for strategy trading like grid trading, arbitrage, day trading, etc.

#### Decreased swap slippage

With much thicker liquidity around the current price point, swap slippage has a notable decrease.

As mentioned above, we've compared our development features of this model in comparison to Uniswaps model, from Ethereum to NEAR, but far less costly. **Ref V2 provides a more capital efficient, user-friendly experience with dynamic pricing, reduced fees, and cross-chain integrations.**

The combination of concentrated liquidity, enhanced limit order and decreased swap slippage offers a good tool kit for trading in NEAR ecosystem. Enjoy it.

### DCL pool

#### getDCLPoolId

Get DCL pool id by tokenA, tokenB and fee.

Note: the fee should be in one of [100, 400, 2000, 10000], which means we charge [0.01%, 0.04%, 0.2%, 1%] separately.

Parameters

```plain
(tokenA:string, tokenB:string, fee:number)
```

Example

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 2000

const pool_id = getDCLPoolId(tokenA, tokenB, fee)
```

```

Response

```
"usdt.fakes.testnet|wrap.testnet|2000"

#### listDCLPools

List all DCL pools

Parameters

```plain
None
```

Example

```plain
const allDCLPools = await listDCLPools()
```

Response

```
[
{
pool_id: 'usdt.fakes.testnet|wrap.testnet|100',
token_x: 'usdt.fakes.testnet',
token_y: 'wrap.testnet',
fee: 100,
point_delta: 1,
current_point: 391459,
liquidity: '0',
liquidity_x: '0',
max_liquidity_per_point: '212676346402870037870835460372692',
volume_x_in: '0',
volume_y_in: '0',
volume_x_out: '0',
volume_y_out: '0',
total_liquidity: '0',
total_order_x: '1000000000000',
total_order_y: '0',
total_x: '1000000000000',
total_y: '0',
state: 'Running'
},
...
]
```

#### getDCLPool

Get DCL pool by pool id

Parameters

```plain
(pool_id: string)
```

Example

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 2000

const pool_id = getDCLPoolId(tokenA, tokenB, fee)

const pool = await getDCLPool(pool_id)

```

Response

```
{
pool_id: 'usdt.fakes.testnet|wrap.testnet|2000',
token_x: 'usdt.fakes.testnet',
token_y: 'wrap.testnet',
fee: 2000,
point_delta: 40,
current_point: 380120,
liquidity: '1110725876876975',
liquidity_x: '1110725876876975',
max_liquidity_per_point: '85068465018609150637077724914819 18',
volume_x_in: '8754783773',
volume_y_in: '1097482248276676850312166 06',
volume_x_out: '1220525924',
volume_y_out: '1301582127053053120024924721',
total_liquidity: '9961756656973511',
total_order_x: '101475486843',
total_order_y: '319070473969235368421944 49',
total_x: '125636676594',
total_y: '1316453492543036053411042 69',
state: 'Running'
}
```

### DCL Swap

#### quote

quote output amount by pool_ids, input_amount, input_token, output_token

Parameters

```plain
{
pool_ids: string[];
input_token: TokenMetadata;
output_token: TokenMetadata;
input_amount: string;
tag?: string;
}
```

Example

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";
const fee = 10000

const pool_ids = [getDCLPoolId(tokenA, tokenB, fee)];

const res = await quote({
pool_ids,
input_amount,
input_token: tokenA,
output_token: tokenB,
});
```

Response

```
{ amount: '20380776164509964256672З', tag: null }
```

#### quote_by_output

quote in put amount by output amount to price by pool_ids, input_amount, input_token, output_token

Parameters

```plain
{
pool_ids: string[];
input_token: TokenMetadata;
output_token: TokenMetadata;
input_amount: string;
tag?: string;
}
```

Example

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 10000
const pool_ids = [getDCLPoolId(tokenA, tokenB, fee)];

const res = await quote_by_output({
pool_ids,
output_amount: "0.1",
input_token: tokenA,
output_token: tokenB,
});
```

Response

```
{ amount: '490370', tag: null }
```

#### DCLSwap

This function integrates Swap, SwapByOutput and LimitOrderWithSwap apis.

- Swap: swap from tokenA to get tokenB.
- SwapByOutput: Swap to get specific tokenB and comsume tokenA.
- LimitOrderWithSwap: compare the calculated price based on parameters with price in the pool, if calculated price is lower than the pool price, go to Swap, the left will generate an limit order higher than the pool price, and if no amount is left, no order generated; if the calculated price is higher than the pool price, will directly generate an limit order.

Parameters

```
interface SwapInfo {
tokenA: TokenMetadata;
tokenB: TokenMetadata;
amountA: string;
}

interface DCLSwapProps {
swapInfo: SwapInfo;
Swap?: {
pool_ids: string[];
min_output_amount: string;
};
SwapByOutput?: {
pool_ids: string[];
output_amount: string;
};
LimitOrderWithSwap?: {
pool_id: string;
output_amount: string;
};
AccountId: string;
}
```

Example (Swap)

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 2000

const pool_ids = [getDCLPoolId(tokenA, tokenB, fee)];

const res = await DCLSwap({
swapInfo: {
amountA: input_amount,
tokenA: tokenA,
tokenB: tokenB,
},
Swap: {
min_output_amount: "0",
pool_ids,
},
AccountId,
});
```

Response (Swap)

```
[
{
receiverId: "usdt.fakes.testnet",
functionCalls: [
{
methodName: "ft_transfer_call",
args: {
receiver_id: "dcl.ref-dev.testnet",
amount: "1000000",
msg: '{"Swap":{"pool_ids":["usdt.fakes.testnet|wrap.testnet|
2000"],"output_token":"wrap.testnet","min_output_amount":"0"}}',
},
gas: "180000000000000",
amount: "0.000000000000000000000001",
},
],
},
];
```

Example (SwapByOutput)

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 2000

const pool_ids = [getDCLPoolId(tokenA, tokenB, fee)];

const res = await DCLSwap({
swapInfo: {
amountA: input_amount,
tokenA: tokenA,
tokenB: tokenB,
},
SwapByOutput: {
pool_ids,
output_amount: "4.89454792",
},
AccountId,
});
```

Response (SwapByOutput)

```
```

```
[
{
receiverId: "usdt.fakes.testnet",
functionCalls: [
{
methodName: "ft_transfer_call",
args: {
receiver_id: "dcl.ref-dev.testnet",
amount: "900000",
msg: '{"SwapByOutput":{"pool_ids":["usdt.fakes.testnet|wrap.testnet|2000"],"output_token":"wrap.testnet","output_amount":"489454792000000000000000"}}',
},
gas: "180000000000000",
amount: "0.000000000000000000000001",
},
],
},
];
```

Example (LimitOrderWithSwap)

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const fee = 2000

const pool_ids = [getDCLPoolId(tokenA, tokenB, fee)];

const res = await DCLSwap({
swapInfo: {
amountA: input_amount,
tokenA: tokenA,
tokenB: tokenB,
},
LimitOrderWithSwap: {
pool_id,
output_amount: "3217.929",
},
AccountId,
});
```

```

Response (LimitOrderWithSwap)

```
[
{
receiverId: "usdt.fakes.testnet",
functionCalls: [
{
methodName: "ft_transfer_call",
args: {
receiver_id: "dcl.ref-dev.testnet",
amount: "1000000",
msg: '{"LimitOrderWithSwap":{"pool_id":"usdt.fakes.testnet|wrap.testnet|2000","buy_token":"wrap.testnet","point":495240}}',
},
gas: "180000000000000",
amount: "0.000000000000000000000001",
},
],
},
];

#### DCLSwapByInputOnBestPool

This function helps to swap on best price pool of at most 4 candidate pools based on tokenA, tokenB, amountA (input amount) and slippageTolerance.

Parameters

```plain
{
tokenA: TokenMetadata;
tokenB: TokenMetadata;
amountA: string;
slippageTolerance: number;
AccountId: string;
}
```

Example

```plain
const tokenA = "usdt.fakes.testnet";

const tokenB = "wrap.testnet";

const res = await DCLSwapByInputOnBestPool({
tokenA,
tokenB,
amountA: "1",
slippageTolerance: 0.1,
AccountId,
});
```

Response

```
[
{
receiverId: "usdt.fakes.testnet",
functionCalls: [
{
methodName: "ft_transfer_call",
args: {
receiver_id: "dcl.ref-dev.testnet",
amount: "1000000",
msg: '{"Swap":{"pool_ids":["usdt.fakes.testnet|wrap.testnet|10000"],"output_token":"wrap.testnet","min_output_amount":"203606350172806050000000"}}',
},
gas: "180000000000000",
amount: "0.000000000000000000000001",
},
],
},
];

```

### Order

#### UserOrderInfo (interface)

```
interface UserOrderInfo {
order_id: string;
owner_id: string;
pool_id: string;
point: number;
sell_token: string;
created_at: string; // timestamp when this order created
original_amount: string; // original amount of sell_token
remain_amount: string; // remain amount to be swapped, 0 means a history order
cancel_amount: string; // amount after cancel an active order
original_deposit_amount: string; // the input amount of LimitOrderWithSwap through ft_transfer_call, maybe partially into instant Swap
swap_earn_amount: string;   // earn token amount through swap before actual place order
buy_token: string;
unclaimed_amount: string;
bought_amount: string; // accumalated amount you get
}
```

#### list_active_orders

Get your active orders

Parameters

```plain
(AccountId: string)
```

Example

```plain
const res = await list_active_orders(AccountId)
```

Response

```
[
{
```

```
order_id: "usdt.fakes.testnet|wrap.testnet|10000#93",
owner_id: "your-account-id.testnet",
pool_id: "usdt.fakes.testnet|wrap.testnet|10000",
point: 398600,
sell_token: "wrap.testnet",
buy_token: "usdt.fakes.testnet",
original_deposit_amount: "1000000000000000000000000",
swap_earn_amount: "0",
original_amount: "1000000000000000000000000",
cancel_amount: "0",
created_at: "1667292669983952215",
remain_amount: "1000000000000000000000000",
bought_amount: "0",
unclaimed_amount: "0",
},
];
```

#### list_history_orders

Get your history orders

Parameters

```plain
(AccountId: string)
```

Example

```plain
const res = await list_history_orders(AccountId)
```

Response

```
[
{
order_id: "usdc.fakes.testnet|wrap.testnet|2000#47",
owner_id: "juaner.testnet",
pool_id: "usdc.fakes.testnet|wrap.testnet|2000",
point: 399120,
sell_token: "wrap.testnet",
buy_token: "usdc.fakes.testnet",
original_deposit_amount: "1000000000000000000000000",
swap_earn_amount: "0",
original_amount: "1000000000000000000000000",
cancel_amount: "1000000000000000000000000",
created_at: "1665928620632469264",
remain_amount: "0",
bought_amount: "0",
unclaimed_amount: null,
},
];
```

#### cancel_order

cancel an active order

Parameters

```plain
(order_id: string)
```

Example

```plain
const res = await cancel_order("usdt.fakes.testnet|wrap.testnet|10000#93")
```

Response

```
[
{
receiverId: "dcl.ref-dev.testnet",
functionCalls: [
{
methodName: "cancel_order",
args: {
order_id: "usdt.fakes.testnet|wrap.testnet|10000#93",
amount: "1000000000000000000000000",
},
gas: "180000000000000",
},
],
},
];
```

#### claim_order

Claim your unclaimed_amount in the order.

Parameters

```plain
(order_id: string)
```

Example

```plain
const res = await claim_order("usdt.fakes.testnet|wrap.testnet|10000#93")
```

Response

```
[
{
receiverId: "dcl.ref-dev.testnet",
functionCalls: [
{
methodName: "cancel_order",
args: {
order_id: "usdt.fakes.testnet|wrap.testnet|10000#93",
amount: "0",
},
gas: "180000000000000",
},
],
},
```

];

#### get_order

Get order information by order_id

Parameters

```plain
(order_id: string)
```

Example

```plain
const res = await get_order("usdt.fakes.testnet|wrap.testnet|10000#93")
```

Response

```
{
  order_id: 'usdt.fakes.testnet|wrap.testnet|10000#93',
  owner_id: 'juaner.testnet',
  pool_id: 'usdt.fakes.testnet|wrap.testnet|10000',
  point: 398600,
  sell_token: 'wrap.testnet',
  buy_token: 'usdt.fakes.testnet',
  original_deposit_amount: '1000000000000000000000000',
  swap_earn_amount: '0',
  original_amount: '1000000000000000000000000',
  cancel_amount: '0',
  created_at: '1667292669983952215',
  remain_amount: '1000000000000000000000000',
  bought_amount: '0',
  unclaimed_amount: '0'
}
```

### Asset

#### list_user_assets

Get user assets.

Parameters

```plain
(AccountId: string)
```

Example

```plain
const res = await list_user_assets("your-account-id.testnet")
```

Response

```
{
'meta-v2.pool.testnet': '1000000000000000000000000',
'ref.fakes.testnet': '54795238623455655224444',
'usdc.fakes.testnet': '204436562555',
'eth.fakes.testnet': '112348196732799990900',
'usdt.fakes.testnet': '164044235810',
'wrap.testnet': '1004944616890200800000000'
}
```

### Utils

#### priceToPoint

Get point based on input price

Parameters

```plain
{
tokenA: TokenMetadata;
tokenB: TokenMetadata;
```

amountA: string;
amountB: string;
fee: number;
}

Example

```plain
const tokenA = await ftGetTokenMetadata("usdt.fakes.testnet");

const tokenB = await ftGetTokenMetadata("wrap.testnet");

const amountA = "0.9";

const amountB = "4.89454792";

const fee = 400;

const res = priceToPoint({
tokenA,
tokenB,
amountA,
amountB,
fee,
});
```

Response

```
431416
```

#### pointToPrice

Get price from tokenA to tokenB based on point

Parameters

```plain
{
tokenA: TokenMetadata;
tokenB: TokenMetadata;
point: number;
}
```

Example

```plain
const tokenA = await ftGetTokenMetadata("usdt.fakes.testnet");

const tokenB = await ftGetTokenMetadata("wrap.testnet");

const res = pointToPrice({
tokenA,
tokenB,
point: 431416,
});
```

Response

```
5.43528191708623
```

###