Overview

Github: https://github.com/Se7en-Seas/boring-vault-ui

NPM Package: https://www.npmjs.com/package/boring-vault-ui

Arbitrum example with a **delayed withdrawer**:
https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/examples/v2.tsx

BoringVaultV1Provider

Interactions to and from a boring vault are done through a parent 'context/provider' that defines and sets up all relevant functionalities. The library additionally exposes some render components, however only the context is necessary to interact with the vault, within which a fully custom UI may be built.

Inputs

Note all 'Contract' input addresses will be provided by the Coinchange team.

- **vaultContract**: base contract address for the vault
- **tellerContract**: contract address that is responsible for vending user shares/vault tokens
- **accountantContract**: contract address that manages additional state of the vault
- **lensContract**: contract address that exposes readOnly functionalities on the vault
- **ethersProvider**: an ethers provider of your choice to power read and write functionalities for a user/dapp on the vault
- **depositTokens**: a list of accepted deposit tokens in the format
    - **address**: token contract address
    - **decimals**: decimal precision of the token
- **baseToken**: the primary base asset of the vault, in the same token format as denoted as above
- **vaultDecimals**: the decimal precision of the vault itself

Example:

```
import { BoringVaultV1Provider } from 'boring-vault-ui';
import { ethers } from "ethers";

const ethersInfuraProvider = new ethers.InfuraProvider(
  "mainnet",
  process.env.INFURA_API_KEY
```

```
);

<BoringVaultV1Provider
      vaultContract="0xef00B163A04dF9960Eb7D41e40Fc8834589A0677"
      tellerContract="0x772d7DEc5Bb5BD6c328a8bFEf2B34B8fAF819A27"
      accountantContract="0xB1667650501bd29c12308FF9A209885e28960Efc"
  lensContract="0x65bf8AcAac9E7dCeBD5A7b6A50640B0901283d85"
  ethersProvider={ethersInfuraProvider}
  depositTokens={[
    {
      address: "0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48",
      decimals: 6,
    },
  vaultDecimals={18}
>
  {/*
        Your code here
  */}
</BoringVaultV1Provider>
```

isBoringV1ContextReady

This function denotes if the context is ready for usage

Inputs

- None

Outputs

- Boolean denoting if the context is ready to use

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';

const {
  isBoringV1ContextReady,
} = useBoringVaultV1();
```

```
console.warn("Is the Boring Context Ready: ", isBoringV1ContextReady);
```

Vault Metadata

Basic readOnly functions that provide information about the vault. These can be done without having a user's wallet connected and are meant to provide and show basic data about the vault.

fetchTotalAssets

This function retrieves a vault's TVL in terms of the baseAsset of the vault

Inputs

- None

Outputs

- A promise that returns the decimal adjusted (human readable) total asset numerical value of the vault (aka TVL) in terms of the baseAsset

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
  isBoringV1ContextReady,
  fetchTotalAssets,
} = useBoringVaultV1();

const [assets, setAssets] = React.useState<number>(0);
useEffect(() => {
    if (!isBoringV1ContextReady) return;
    fetchTotalAssets().then((assets) => {
      console.log("The Vaults TVL: ", assets);
      setAssets(assets);
    });
  }, [isBoringV1ContextReady]);
```

fetchShareValue

This function provides the value for 1 share of the vault in terms of the underlying baseAsset.

Inputs

- None

Outputs

- A promise that returns the decimal adjusted (human readable) numerical value for 1 share in terms of the underlying baseAsset.

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
  isBoringV1ContextReady,
  fetchShareValue
} = useBoringVaultV1();


const [shareValue, setShareValue] = React.useState<number>(0);
useEffect(() => {
  if (!isBoringV1ContextReady) return;
  fetchShareValue().then((value) => {
    console.log("Share value: ", value);
    setShareValue(value);
  });
}, [isBoringV1ContextReady]);
```

User Metadata

These are functions that requires a user to be connected to the dapp with their address exposed via useAddress from wagmi and provide user specific information.

fetchUserShares

This function provides the decimal adjusted (human readable) numerical value of vault shares that a user owns in their account.

Inputs

- **userAddress**: the address of the user in the vault you'd like to get the shares for

Outputs

- A promise that returns the decimal adjusted (human readable) total numerical value of all shares of a vault a user owns.

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
  isBoringV1ContextReady,
  fetchUserShares
} = useBoringVaultV1();


const [userShares, setUserShares] = React.useState<number>(0);
useEffect(() => {
  if (!isBoringV1ContextReady) return;
  fetchUserShares('0x...').then((shares) => {
    console.log("User shares: ", shares);
    setUserShares(shares);
  });
}, [isBoringV1ContextReady]);
```

fetchUserUnlockTime

This function provides the unlock time for when a user may transfer or withdraw their shares.

Inputs

- **userAddress**: the address of the user in the vault you'd like to get the unlock time for

Outputs

- A promise that returns the Unix Seconds timestamp for when a user may transfer or withdraw their shares from the vault

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
  isBoringV1ContextReady,
  fetchShareValue
} = useBoringVaultV1();


const [userUnlockTime, setUserUnlockTime] = React.useState<number>(0);
useEffect(() => {
  if (!isBoringV1ContextReady) return;
  fetchUserUnlockTime('0x...').then((time) => {
    console.log("User Unlock time: ", time);
    setUserUnlockTime(time);
  });
}, [isBoringV1ContextReady]);
```

Deposits

Functionalities that write to the vault. A comprehensive example may be found here: https://github.com/Se7en-Seas/boring-vault-ui/edit/main/src/components/v1/DepositButton.tsx#L57

deposit

This function checks if a user has approved the spend of an asset into the vault, if not prompts them to do so, and sequentially deposits a user's desired assets into the vault.

Inputs

- **signer**: ****an ethers **JsonRPCSigner**. If you are using viem, you may use this example to create an ethers signer out of a viem wallet client: https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/hooks/ethers.tsx
- **depositAmount**: a decimal adjusted (human readable) **string** that represents the amount of selectedTokens the user wants to deposit into the vault
- **selectedToken**: the **token** a user wants to deposit in the format
    - **address**: token contract address
    - **decimals**: decimal precision of the token

Outputs

- A promise that returns a DepositStatus
    - **initiated**: boolean representing if the deposit function has been called and is in progress of being executed
    - **loading**: boolean representing if there is a relevant deposit transaction ongoing (e.g. approval and/or deposit)
    - **success** (optional): boolean representing if the deposit action succeeded
    - **error** (optional): string representing why a deposit failed (e.g. insufficient balance, approval rejected, etc.)
    - **tx_hash** (optional): the string of a successful deposit transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
    deposit,
  } = useBoringVaultV1();

/*
Definitions for your signer, deposits, and token
*/

deposit(signer!, depositAmount, selectedToken)}
```

depositStatus

This object provides a depositStatus denoting any attributes to an ongoing deposit.

Inputs

- None

Outputs

- Returns a DepositStatus
  - **initiated**: boolean representing if the deposit function has been called and is in progress of being executed
  - **loading**: boolean representing if there is a relevant deposit transaction ongoing (e.g. approval and/or deposit)
  - **success** (optional): boolean representing if the deposit action succeeded
  - **error** (optional): string representing why a deposit failed (e.g. insufficient balance, approval rejected, etc.)
  - **tx_hash** (optional): the string of a successful deposit transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import { useEffect } from "react";
import { useToast } from "@chakra-ui/react";

const {
  depositStatus
} = useBoringVaultV1();

const toast = useToast();
useEffect(() => {
  if (depositStatus.loading) {
    toast({
      title: "Processing deposit...",
      status: "info",
      duration: 3000,
      isClosable: true,
    });
  } else if (depositStatus.success) {
    toast({
      title: "Deposit successful",
      // Consider adding a link to etherscan
      description: `Transaction hash: ${depositStatus.tx_hash}`,
      status: "success",
      duration: 5000,
```

```
      isClosable: true,
    });
  } else if (depositStatus.error) {
    toast({
      title: "Failed to deposit",
      description: depositStatus.error,
      status: "error",
      duration: 5000,
      isClosable: true,
    });
  }
}, [depositStatus, toast]);
```

Delayed Withdraws

Delayed withdraws are withdraws where a user must wait a certain amount of time before being able to return and claim their tokens.

delayWithdraw

This function checks if a user has approved the withdraw contract a sufficient amount to transfer vault tokens, if not prompts the user to do so, and finally submits a withdrawal intent.

IMPORTANT NOTE: If this function is called multiple times for a specific token, the amounts will stack/be summed (as opposed to being replaced). It is recommended to disable additional withdrawal intents if one is already ongoing.

Inputs

- **signer**: ****an ethers **JsonRPCSigner**. If you are using viem, you may use this example to create an ethers signer out of a viem wallet client: https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/hooks/ethers.tsx
- **shareAmount**: a decimal adjusted (human readable) **string** that represents the amount of vault shares a user wants to withdraw
- **tokenOut**: the **token** the user wants to receive
    - **address**: token contract address
    - **decimals**: decimal precision of the token
- **maxLoss**: a human readable percent (e.g. 1 = 1%) as a string that represents the max deviation from the share price the user is willing to accept. If the share price deviates by

more than this percent in either direction (up/down) the withdraw becomes invalid.
    - ***IMPORTANT NOTE: If this is set to 0, the default value for the contract will be used***
- **thirdPartyClaimer**: a boolean representing if the user wants to allow anyone to finish their withdraw for them. A user will still receive their tokens, it simply indicates that someone else can initiate the claim intent on their behalf.

Outputs

- A promise that returns a WithdrawStatus
    - **initiated**: boolean representing if the withdraw function has been called and is in progress of being executed
    - **loading**: boolean representing if there is a relevant withdraw transaction ongoing (e.g. approval and/or withdraw)
    - **success** (optional): boolean representing if the withdraw intent action succeeded
    - **error** (optional): string representing why a withdraw failed (e.g. insufficient balance, approval rejected, etc.)
    - **tx_hash** (optional): the string of a successful withdraw transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
    delayWithdraw,
  } = useBoringVaultV1();

/*
Definitions for your signer, shareAmount, tokenOut, maxLoss,
and thirdPartyClaimer variables
*/

delayWithdraw(signer!, shareAmount, tokenOut, maxLoss, thirdPartyClaimer)}
```

delayWithdrawStatuses

This function retrieves a list of all non-zero withdraw intents.

Inputs

- **signer**: ****an ethers **JsonRPCSigner**. If you are using viem, you may use this example to create an ethers signer out of a viem wallet client: https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/hooks/ethers.tsx

Outputs

- A promise that returns a list of **DelayWithdrawStatus**es
    - **allowThirdPartyToComplete**: boolean if another account may complete the withdraw intent on behalf of the user
    - **maxLoss**: number representing the max share value deviation the request is willing to accept (e.g. 1 = 1%)
    - **maturity**: a unix timestamp (number) that indicates when the request may be completed/output tokens claimed
    - **shares**: human readable number of how many shares the user indicated they wanted to withdraw
    - **exchangeRateAtTimeOfRequest**: the human readable exchange rate that 1 share may have been redeemed for at the time of the request (this is mostly informational and does not need to be considered in the user render/ux flow)
    - **token:** the output token of the request
        - **address**: token contract address
        - **decimals**: decimal precision of the token

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";
// Custom viem to ethers hook (example above in 'Inputs')
import { useEthersSigner } from "../../hooks/ethers";

const {
    delayWithdrawStatuses,
  } = useBoringVaultV1();

const [statuses, setStatuses] = useState<any[]>([]);
const signer = useEthersSigner();

useEffect(() => {
  const fetchStatuses = async () => {
    const fetchedStatuses = await delayWithdrawStatuses(signer!);
    setStatuses(fetchedStatuses);
  };
```

```
  fetchStatuses();
}, [delayWithdrawStatuses, signer]);
```

delayWithdrawCancel

This function cancels a user's withdrawal request for a given output asset.

Inputs

- **signer**: ****an ethers **JsonRPCSigner**. If you are using viem, you may use this example to create an ethers signer out of a viem wallet client: https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/hooks/ethers.tsx
- **tokenOut**: the output **token** to cancel the request for
    - **address**: token contract address
    - **decimals**: decimal precision of the token

Outputs

- A promise that returns a WithdrawStatus
    - **initiated**: boolean representing if the withdraw function has been called and is in progress of being executed
    - **loading**: boolean representing if there is a relevant withdraw transaction ongoing
    - **success** (optional): boolean representing if the withdraw intent action succeeded
    - **error** (optional): a string representing why a withdrawal failed
    - **tx_hash** (optional): the string of a successful withdrawal transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
    delayWithdrawCancel,
  } = useBoringVaultV1();

/*
Definitions for your signer and tokenOut variables
*/

delayWithdrawCancel(signer!, tokenOut)}
```

delayWithdrawComplete

This function completes a withdrawal for a given asset and transfers the output tokens to the user that created the original withdrawal intent.

***Important Note**: Your UX should validate that the request's maturity is greater than or equal to the current time. If this is not the case at the time of calling this function, the execution will revert.*

Inputs

- **signer**: ****an ethers **JsonRPCSigner**. If you are using viem, you may use this example to create an ethers signer out of a viem wallet client: https://github.com/Se7en-Seas/boring-vault-ui/blob/main/src/hooks/ethers.tsx
- **tokenOut**: the output **token** of the specific withdrawal intent
    - **address**: token contract address
    - **decimals**: decimal precision of the token

Outputs

- A promise that returns a WithdrawStatus
    - **initiated**: boolean representing if the withdraw function has been called and is in progress of being executed
    - **loading**: boolean representing if there is a relevant withdraw transaction ongoing
    - **success** (optional): boolean representing if the withdraw intent action succeeded
    - **error** (optional): a string representing why a withdrawal failed
    - **tx_hash** (optional): the string of a successful withdrawal transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import React, { useEffect } from "react";

const {
    delayWithdrawComplete,
  } = useBoringVaultV1();

/*
Definitions for your signer and tokenOut variables
*/

delayWithdrawComplete(signer!, tokenOut)}
```

withdrawStatus

This object provides a withdrawStatus denoting any attributes to an ongoing withdraw intent (start/cancel/claim) for any withdraw action (delayed or queued)

Inputs

- None

Outputs

- Returns a WithdrawStatus
    - **initiated**: boolean representing if a withdraw function has been called and is in progress of being executed
    - **loading**: boolean representing if there is a relevant withdraw intent transaction ongoing (e.g. approval/deposit/claim/cancel)
    - **success** (optional): boolean representing if the withdrawal intent succeeded
    - **error** (optional): a string representing why a withdrawal intent failed (e.g. insufficient balance, approval rejected, etc.)
    - **tx_hash** (optional): the string of a successful withdraw intent transaction hash

Example:

```
import { useBoringVaultV1 } from 'boring-vault-ui';
import { useEffect } from "react";
import { useToast } from "@chakra-ui/react";

const {
  withdrawStatus
} = useBoringVaultV1();

const toast = useToast();
useEffect(() => {
  if (withdrawStatus.loading) {
    toast({
      title: "Processing withdraw intent...",
      status: "info",
      duration: 3000,
      isClosable: true,
    });
  } else if (withdrawStatus.success) {
    toast({
      title: "Withdraw Intent successful",
```

```
      // Consider adding a link to etherscan
      description: `Transaction hash: ${withdrawStatus.tx_hash}`,
      status: "success",
      duration: 5000,
      isClosable: true,
    });
  } else if (withdrawStatus.error) {
    toast({
      title: "Failed to complete withdraw intent",
      description: withdrawStatus.error,
      status: "error",
      duration: 5000,
      isClosable: true,
    });
  }
}, [withdrawStatus, toast]);
```