Client App with Sui TypeScript SDK

This exercise diverges from the example built in the previous topics in this section. Rather than adding a frontend to the running example, the instruction walks you through setting up dApp Kit in a React App, allowing you to connect to wallets, and query data from Sui RPC nodes to display in your app. You can use it to create your own frontend for the example used previously, but if you want to get a fully functional app up and running quickly, run the following command in a terminal or console to scaffold a new app with all steps in this exercise already implemented:

You must use the pnpm or yarn package managers to create Sui project scaffolds. Follow the pnpm install or yarn install instructions, if needed.

or

The Sui TypeScript SDK (@mysten/sui) provides all the low-level functionality needed to interact with Sui ecosystem from TypeScript. You can use it in any TypeScript or JavaScript project, including web apps, Node.js apps, or mobile apps written with tools like React Native that support TypeScript.

For more information on the Sui TypeScript SDK, see the Sui TypeScript SDK documentation .

dApp Kit (@mysten/dapp-kit) is a collection of React hooks, components, and utilities that make building dApps on Sui straightforward. For more information on dApp Kit, see the dApp Kit documentation .

To get started, you need a React app. The following steps apply to any React, so you can follow the same steps to add dApp Kit to an existing React app. If you are starting a new project, you can use Vite to scaffold a new React app.

Run the following command in your terminal or console, and select React as the framework, and then select one of the TypeScript templates:

Now that you have a React app, you can install the necessary dependencies to use dApp Kit:

To use all the features of dApp Kit, wrap your app with a couple of Provider components.

Open the root component that renders your app (the default location the Vite template uses is src/main.tsx) and integrate or replace the current code with the following.

The first Provider to set up is the QueryClientProvider from @tanstack/react-query . This Provider manages request state for various hooks in dApp kit. If you're already using @tanstack/react-query , dApp Kit can share the same QueryClient instance.

Next, set up the SuiClientProvider . This Provider delivers a SuiClient instance from @mysten/sui to all the hooks in dApp Kit. This provider manages which network dApp Kit connects to, and can accept configuration for multiple networks. This exercise connects to devnet .

Finally, set up the WalletProvider from @mysten/dapp-kit , and import styles for the dapp-kit components.

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

What is the Sui TypeScript SDK?

The Sui TypeScript SDK (@mysten/sui) provides all the low-level functionality needed to interact with Sui ecosystem from

TypeScript. You can use it in any TypeScript or JavaScript project, including web apps, Node.js apps, or mobile apps written with tools like React Native that support TypeScript.

For more information on the Sui TypeScript SDK, see the Sui TypeScript SDK documentation .

dApp Kit (@mysten/dapp-kit) is a collection of React hooks, components, and utilities that make building dApps on Sui straightforward. For more information on dApp Kit, see the dApp Kit documentation .

To get started, you need a React app. The following steps apply to any React, so you can follow the same steps to add dApp Kit to an existing React app. If you are starting a new project, you can use Vite to scaffold a new React app.

Run the following command in your terminal or console, and select React as the framework, and then select one of the TypeScript templates:

Now that you have a React app, you can install the necessary dependencies to use dApp Kit:

To use all the features of dApp Kit, wrap your app with a couple of Provider components.

Open the root component that renders your app (the default location the Vite template uses is src/main.tsx) and integrate or replace the current code with the following.

The first Provider to set up is the QueryClientProvider from @tanstack/react-query . This Provider manages request state for various hooks in dApp kit. If you're already using @tanstack/react-query , dApp Kit can share the same QueryClient instance.

Next, set up the SuiClientProvider . This Provider delivers a SuiClient instance from @mysten/sui to all the hooks in dApp Kit. This provider manages which network dApp Kit connects to, and can accept configuration for multiple networks. This exercise connects to devnet .

Finally, set up the WalletProvider from @mysten/dapp-kit , and import styles for the dapp-kit components.

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

What is dApp Kit?

dApp Kit (@mysten/dapp-kit) is a collection of React hooks, components, and utilities that make building dApps on Sui straightforward. For more information on dApp Kit, see the dApp Kit documentation .

To get started, you need a React app. The following steps apply to any React, so you can follow the same steps to add dApp Kit to an existing React app. If you are starting a new project, you can use Vite to scaffold a new React app.

Run the following command in your terminal or console, and select React as the framework, and then select one of the TypeScript templates:

Now that you have a React app, you can install the necessary dependencies to use dApp Kit:

To use all the features of dApp Kit, wrap your app with a couple of Provider components.

Open the root component that renders your app (the default location the Vite template uses is src/main.tsx) and integrate or replace the current code with the following.

The first Provider to set up is the QueryClientProvider from @tanstack/react-query . This Provider manages request state for

various hooks in dApp kit. If you're already using @tanstack/react-query , dApp Kit can share the same QueryClient instance.

Next, set up the SuiClientProvider . This Provider delivers a SuiClient instance from @mysten/sui to all the hooks in dApp Kit. This provider manages which network dApp Kit connects to, and can accept configuration for multiple networks. This exercise connects to devnet .

Finally, set up the WalletProvider from @mysten/dapp-kit , and import styles for the dapp-kit components.

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Installing dependencies

To get started, you need a React app. The following steps apply to any React, so you can follow the same steps to add dApp Kit to an existing React app. If you are starting a new project, you can use Vite to scaffold a new React app.

Run the following command in your terminal or console, and select React as the framework, and then select one of the TypeScript templates:

Now that you have a React app, you can install the necessary dependencies to use dApp Kit:

To use all the features of dApp Kit, wrap your app with a couple of Provider components.

Open the root component that renders your app (the default location the Vite template uses is src/main.tsx) and integrate or replace the current code with the following.

The first Provider to set up is the QueryClientProvider from @tanstack/react-query . This Provider manages request state for various hooks in dApp kit. If you're already using @tanstack/react-query , dApp Kit can share the same QueryClient instance.

Next, set up the SuiClientProvider . This Provider delivers a SuiClient instance from @mysten/sui to all the hooks in dApp Kit. This provider manages which network dApp Kit connects to, and can accept configuration for multiple networks. This exercise connects to devnet .

Finally, set up the WalletProvider from @mysten/dapp-kit , and import styles for the dapp-kit components.

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Setting up Provider components

To use all the features of dApp Kit, wrap your app with a couple of Provider components.

Open the root component that renders your app (the default location the Vite template uses is src/main.tsx) and integrate or replace the current code with the following.

The first Provider to set up is the QueryClientProvider from @tanstack/react-query . This Provider manages request state for various hooks in dApp kit. If you're already using @tanstack/react-query , dApp Kit can share the same QueryClient instance.

Next, set up the SuiClientProvider . This Provider delivers a SuiClient instance from @mysten/sui to all the hooks in dApp Kit. This provider manages which network dApp Kit connects to, and can accept configuration for multiple networks. This exercise connects to devnet .

Finally, set up the WalletProvider from @mysten/dapp-kit , and import styles for the dapp-kit components.

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Connecting to a wallet

With all Providers set up, you can use dApp Kit hooks and components. To allow users to connect their wallets to your dApp, add a ConnectButton .

The ConnectButton component displays a button that opens a modal on click, enabling the user to connect their wallet. Upon connection, it displays their address, and provides the option to disconnect.

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Getting the connected wallet address

Now that you have a way for users to connect their wallets, you can start using the useCurrentAccount hook to get details about the connected wallet account.

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Querying data from Sui RPC nodes

Now that you have the account to connect to, you can query for objects the connected account owns:

You now have a dApp connected to wallets and can query data from RPC nodes.

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.

Related links

The next step from here is to start interacting with Move modules, constructing transaction blocks, and making Move calls. This exercise continues in the Counter end-to-end example.