Swap Guide

This guide will walk you through creating a basic token swap app using the STON.fi SDK and API in a **React** project. We'll integrate wallet connectivity with **TonConnect** (via `@tonconnect/ui-react`) to allow users to connect their TON wallet and perform a swap. The guide is beginner-friendly and assumes minimal React experience.

> **Note**: In this demo, we will leverage **Tailwind CSS** for styling instead of using custom CSS. The setup for Tailwind CSS is already included in the instructions below, so you don't need to set it up separately.

> **Note**: You can use any package manager (npm, yarn, pnpm, or bun) to set up your React project. In this tutorial, we'll demonstrate with **pnpm**.

Table of Contents

---

1. Introduction

In this quickstart, we will build a minimal React app to:

- Connect to a TON wallet (via **TonConnect UI**).
- Fetch available tokens from STON.fi (via `@ston-fi/api`).
- Simulate a swap (to see expected output).
- Execute a swap transaction on-chain (via `@ston-fi/sdk`).

We will use:

- `@ston-fi/sdk` – Helps build the payload for the actual swap transaction.
- `@ston-fi/api` – Lets us fetch asset lists, pool data, and run swap simulations.
- `@tonconnect/ui-react` – Provides a React-based TON wallet connect button and utilities.

---

2. Setting Up the Project

2.1 Create a React App

First, let's check if pnpm is installed on your system:

```
pnpm --version
```

If you see a version number (like `10.4.0`), pnpm is installed. If you get an error, you'll need to install pnpm first:

```
npm install -g pnpm
```

Now we'll create a new React project using **Vite**. However, you can use any React setup you prefer (Next.js, CRA, etc.).

Run the following command to create a new Vite-based React project:

```
pnpm create vite --template react
```

When prompted, type your desired project name (e.g., stonfi-swap-app):

```
Project name: » stonfi-swap-app
```

Then enter the folder:

```
cd stonfi-swap-app
```

2.2 Installing the Required Packages

Within your new React project directory, install the STON.fi packages, TonConnect UI, and the TON SDK:

```
pnpm add @ston-fi/sdk @ston-fi/api @tonconnect/ui-react @ton/ton
```

Next, install Tailwind CSS and its Vite plugin:

```
pnpm add tailwindcss @tailwindcss/vite
```

Additionally, install the Node.js polyfills plugin for Vite, which is necessary to provide Buffer and other Node.js APIs in the browser environment (required by TON libraries):

```
pnpm add vite-plugin-node-polyfills
```

Configure the Vite plugin by updating `vite.config.js` file:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'
import { nodePolyfills } from 'vite-plugin-node-polyfills'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss(), nodePolyfills()],
})
```

Then, import Tailwind CSS in your main CSS file. Open `src/index.css` and replace all code with:

```
@import "tailwindcss";
```

You can also remove `src/App.css` we don't need it anymore, and remove the import statement `import './App.css'` from `src/App.jsx`.

After making these changes, you can verify that your app still runs correctly by starting the development server:

```
pnpm install
pnpm dev
```

This should launch your app in development mode, typically at `http://localhost:5173`. You should see the Vite + React logo and text on a plain white background. Since we've removed the default styling (App.css), the page will look simpler than the default template.

If you see the logo and text, it means your Vite + React setup is working correctly. Make sure everything loads without errors before proceeding to the next step.

---

3. Connecting the Wallet

3.1 Add the TonConnect Provider

Open **src/main.jsx** (Vite's default entry point) and wrap your application with the `TonConnectUIProvider`. This provider makes the TonConnect context available to your app for wallet connectivity. Also, point it to a manifest file (which we will create next) that describes your app to wallets.

```
// src/main.jsx
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import { TonConnectUIProvider } from '@tonconnect/ui-react';
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <TonConnectUIProvider manifestUrl=
{`${window.location.origin}/tonconnect-manifest.json`}>
      <App />
    </TonConnectUIProvider>
  </StrictMode>,
)
```

This wraps the `<App />` component with `TonConnectUIProvider`. The `manifestUrl` points to a `tonconnect-manifest.json` file that should be served at the root of your app (we'll create this below). Using `window.location.origin` ensures it picks the correct host (e.g., `http://localhost:5173/tonconnect-manifest.json` in development).

3.2 Create the TonConnect Manifest

In the **public** folder of your project, create a file named **tonconnect-manifest.json**. This manifest provides wallet apps with information about your application (like name and icon). You should customize this manifest for your own application. Here's an example:

```
{
    "url": "https://ton.vote",
    "name": "TON Vote",
    "iconUrl": "https://ton.vote/logo.png"
}
```

Make sure to update these fields for your application:

- **url**: The base URL where your app is served
- **name**: Your application's display name (this is what wallets will show to users)
- **iconUrl**: A link to your app's icon (should be a 180×180 PNG image)

Make sure this file is accessible. When the dev server runs, you should be able to fetch it in your browser at `http://localhost:5173/tonconnect-manifest.json`.

3.3 Add the Connect Wallet Button

In your main **App** component (e.g., **src/App.jsx**), import and include the `TonConnectButton`. For example:

```
// src/App.jsx
import { TonConnectButton } from '@tonconnect/ui-react';

function App() {
  return (
    <div className="flex flex-col items-center justify-center min-h-
screen p-4">
      <h1 className="text-2xl font-bold mb-4">STON.fi Swap Demo</h1>
      <TonConnectButton />
    </div>
  );
}

export default App;
```

This will render a "Connect Wallet" button. When clicked, it opens a modal with available TON wallets. After the user connects, the button will automatically display the wallet address or account info. TonConnect UI handles the connection state for you.

4. Fetching Available Assets

Next, let's retrieve the list of tokens (assets) that can be swapped on STON.fi. We use the STON.fi API client (`@ston-fi/api`) for this. It's important to note that while many tokens on the TON blockchain use a decimal precision of 9, some tokens, like jUSDT, have a decimal precision of 6. Therefore, we rely on the token's metadata to dynamically determine the decimal precision, ensuring compatibility across different tokens.

1. Initialize the API client: In the component where you will handle swapping (we can continue working in **App.jsx** for simplicity), create a client instance from `StonApiClient`. This has methods to get assets, pools, simulate swaps, etc.
2. Fetch assets on load: Using React's effect hook, fetch the asset list when the component mounts. Store the assets in state so you can display them.

```
// src/App.jsx
import { useEffect, useState } from 'react';
import { TonConnectButton } from '@tonconnect/ui-react';
import { StonApiClient, AssetTag } from '@ston-fi/api';

function App() {
  const [assets, setAssets] = useState([]);
  const [fromAsset, setFromAsset] = useState(null);
  const [toAsset, setToAsset] = useState(null);
  const [amount, setAmount] = useState(0);

  // Single function to handle changes in "From", "To", and "Amount"
  // Clears the simulation result each time any input changes
  const handleChange = (setter) => (e) => {
    const value = e.target.value;

    if (setter === setFromAsset || setter === setToAsset) {
      const selectedAsset = assets.find(asset => asset.contractAddress
=== value);
      setter(selectedAsset);
    } else {
      setter(value);
    }
  };

  // Helper to find an asset by address and return a consistent object
  const getAssetInfo = (asset) => {
    if (!asset) return { symbol: 'token', decimals: 10 ** 9 };

    // Determine display symbol
    const symbol = asset.meta?.symbol || asset.meta?.displayName ||
'token';

    // Always take the decimal property from metadata, fallback to 9 if
missing
    const decimals = 10 ** (asset.meta?.decimals ?? 9);

    return { symbol, decimals };
  };

  useEffect(() => {
    const fetchAssets = async () => {
      try {
        const client = new StonApiClient();
        const condition = [
          AssetTag.LiquidityVeryHigh,
          AssetTag.LiquidityHigh,
          AssetTag.LiquidityMedium,
        ].join(' | ');
        const assetList = await client.queryAssets({ condition });

        setAssets(assetList);
        if (assetList[0]) setFromAsset(assetList[0]);
        if (assetList[1]) setToAsset(assetList[1]);
      } catch (err) {
        console.error('Failed to fetch assets:', err);
      }
    };
    fetchAssets();
  }, []);

  // Shortcut to display either symbol or 'token'
  const displaySymbol = (asset) => getAssetInfo(asset).symbol;

  return (
    <div className="flex flex-col items-center justify-center min-h-
screen bg-gradient-to-b from-blue-50 to-indigo-100 p-6">
      <div className="w-full max-w-md bg-white rounded-xl shadow-lg p-6
space-y-6">
        <div className="flex justify-between items-center">
          <h1 className="text-3xl font-bold text-indigo-700">STON.fi
Swap</h1>
          <TonConnectButton />
        </div>

        <div className="h-px bg-gray-200 w-full my-4"></div>

        {assets.length > 0 ? (
          <div className="space-y-6">
            {/* From */}
            <div className="flex flex-col">
              <label className="text-sm font-medium text-gray-600 mb-
1">
                From
              </label>
              <select
                value={fromAsset?.contractAddress || ''}
                onChange={handleChange(setFromAsset)}
                className="w-full p-3 bg-gray-50 border border-gray-200
rounded-lg focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500
transition-all"
              >
                {assets.map((asset) => (
                  <option key={asset.contractAddress} value=
{asset.contractAddress}>
                    {asset.meta?.symbol || asset.meta?.displayName ||
'token'}
                  </option>
                ))}
              </select>
            </div>

            {/* To */}
            <div className="flex flex-col">
              <label className="text-sm font-medium text-gray-600 mb-
1">
                To
              </label>
              <select
                value={toAsset?.contractAddress || ''}
                onChange={handleChange(setToAsset)}
                className="w-full p-3 bg-gray-50 border border-gray-200
rounded-lg focus:ring-2 focus:ring-indigo-500 focus:border-indigo-500
transition-all"
              >
                {assets.map((asset) => (
                  <option key={asset.contractAddress} value=
{asset.contractAddress}>
                    {asset.meta?.symbol || asset.meta?.displayName ||
'token'}
                  </option>
                ))}
              </select>
            </div>

            {/* Amount */}
            <div className="flex flex-col">
              <label className="text-sm font-medium text-gray-600 mb-
1">
                Amount
              </label>
              <div className="relative">
                <input
                  type="number"
                  placeholder="0.0"
                  value={amount}
                  onChange={handleChange(setAmount)}
                  className="w-full p-3 bg-gray-50 border border-gray-
200 rounded-lg focus:ring-2 focus:ring-indigo-500 focus:border-indigo-
500 transition-all"
                />
                <div className="absolute right-3 top-1/2 -translate-y-
1/2 text-gray-500 text-sm">
                  {displaySymbol(fromAsset)}
                </div>
              </div>
            </div>

            {/* Buttons */}
            <div className="flex space-x-3">
              <button
                className="flex-1 bg-indigo-100 hover:bg-indigo-200
text-indigo-700 font-medium py-3 px-4 rounded-lg transition-all"
              >
                Simulate
              </button>
              <button className="flex-1 bg-indigo-600 hover:bg-indigo-
700 text-white font-medium py-3 px-4 rounded-lg transition-all">
                Swap
              </button>
            </div>
          </div>
        ) : (
          <div className="flex justify-center items-center py-10">
            <div className="animate-pulse flex space-x-2">
              <div className="h-2 w-2 bg-indigo-500 rounded-full">
</div>
              <div className="h-2 w-2 bg-indigo-500 rounded-full">
</div>
              <div className="h-2 w-2 bg-indigo-500 rounded-full">
</div>
            </div>
            <p className="ml-3 text-gray-600">Loading assets...</p>
          </div>
        )}
      </div>

      <div className="mt-6 text-center text-xs text-gray-500">
        Powered by STON.fi
      </div>
    </div>
  );
}

export default App;
```

5. Simulating a Swap

```
// Step #5 changes in App.jsx
// keep all imports from step #4

function App() {
  // keep existing state from step #4 and add simulationResult state
  const [simulationResult, setSimulationResult] = useState(null);

  // update handleChange from step #4 to reset simulationResult when
changing inputs
  const handleChange = (setter) => (e) => {
  // keep existing code from step #4

    setSimulationResult(null);
  };

  // add the handleSimulate function
  const handleSimulate = async () => {
    if (!fromAsset || !toAsset || !amount) return;
    try {
      const { decimals: fromDecimals } = getAssetInfo(fromAsset);
      const client = new StonApiClient();

      // Convert user-facing amount (e.g. 10.5) to blockchain units
(e.g. 10500000000)
      // by multiplying by token decimals and converting to string for
the API
      const offerUnits = (Number(amount) * fromDecimals).toString();

      const result = await client.simulateSwap({
        offerAddress: fromAsset.contractAddress,
        askAddress: toAsset.contractAddress,
        slippageTolerance: '0.01',
        offerUnits,
      });
      setSimulationResult(result);
    } catch (err) {
      console.error('Simulation failed:', err);
      setSimulationResult(null);
    }
  };

  // Convert blockchain units (e.g. 10500000000) back to user-friendly
format (e.g. 10.5000)
  // This reverses the process done for input, dividing by token
decimals
  const formattedOutputAmount = simulationResult
    ? (Number(simulationResult.minAskUnits) /
getAssetInfo(toAsset).decimals).toFixed(4)
```

```
    : '';

  return (
    // keep the same JSX wrapper from step #4
    <div className="flex flex-col items-center justify-center min-h-
screen bg-gradient-to-b from-blue-50 to-indigo-100 p-6">
        {/* ... existing code from step #4 ... */}

        {/* Update the Simulate button to attach handler */}
        <button
          onClick={handleSimulate}
          className="flex-1 bg-indigo-100 hover:bg-indigo-200 text-
indigo-700 font-medium py-3 px-4 rounded-lg transition-all"
        >
          Simulate
        </button>
        {/* ... existing code from step #4 ... */}

        {/* Show simulation result if present */}

        {/* add simulation result */}
        {simulationResult && (
          <div className="mt-4 w-full max-w-md bg-white rounded-xl
shadow-lg p-4">
            <div className="text-center">
              <p className="text-lg font-medium text-gray-800">Swap
Summary</p>
              <div className="flex justify-center items-center space-x-
2 mt-2">
                <p className="text-md font-bold">
                  {amount} {displaySymbol(fromAsset)}
```

6. Executing a Swap Transaction

```
// Step #6 changes

// add extra imports:
import { dexFactory, Client } from "@ston-fi/sdk";
import { TonConnectButton, useTonConnectUI, useTonAddress } from
'@tonconnect/ui-react';
```

```
            </div>
          </div>
        )}
      <div className="mt-6 text-center text-xs text-gray-500">
        Powered by STON.fi
      </div>
    </div>
  );
}
```

```
// continue in App.jsx
const [tonConnectUI] = useTonConnectUI();
const userAddress = useTonAddress();

  const handleSwap = async () => {
    if (!fromAsset || !toAsset || !amount || !userAddress) {
      alert('Please connect wallet and enter swap details.');
      return;
    }

    if(!simulationResult) {
      alert('Please simulate the swap first.');
      return;
    }

    try {
      // 1. Initialize API client
      const tonApiClient = new Client({
        endpoint: "https://toncenter.com/api/v2/jsonRPC",
      });

      // 2. Get router metadata and create DEX instance
      const client = new StonApiClient();
      const routerMetadata = await
client.getRouter(simulationResult.routerAddress);
      const dexContracts = dexFactory(routerMetadata);
      const router = tonApiClient.open(
        dexContracts.Router.create(routerMetadata.address)
      );

      // 3. Prepare common transaction parameters
      const sharedTxParams = {
        userWalletAddress: userAddress,
        offerAmount: simulationResult.offerUnits,
        minAskAmount: simulationResult.minAskUnits,
      };

      // 4. Determine swap type and get transaction parameters
      const getSwapParams = () => {
        // TON -> Jetton
        if (fromAsset.kind === 'Ton') {
          return router.getSwapTonToJettonTxParams({
            ...sharedTxParams,
            proxyTon:
dexContracts.pTON.create(routerMetadata.ptonMasterAddress),
            askJettonAddress: simulationResult.askAddress,
          });
        }
        // Jetton -> TON
        if (toAsset.kind === 'Ton') {
          return router.getSwapJettonToTonTxParams({
            ...sharedTxParams,
            proxyTon:
dexContracts.pTON.create(routerMetadata.ptonMasterAddress),
            offerJettonAddress: simulationResult.offerAddress,
          });
        }
        // Jetton -> Jetton (no proxyTon needed)
        return router.getSwapJettonToJettonTxParams({
          ...sharedTxParams,
          offerJettonAddress: simulationResult.offerAddress,
          askJettonAddress: simulationResult.askAddress,
        });
      };

      const swapParams = await getSwapParams();

      // 5. Send transaction via TonConnect
      await tonConnectUI.sendTransaction({
        validUntil: Date.now() + 5 * 60 * 1000,
        messages: [
          {
            address: swapParams.to.toString(),
            amount: swapParams.value.toString(),
            payload: swapParams.body?.toBoc().toString("base64"),
          }
        ]
      });
    } catch (err) {
      console.error('Swap failed:', err);
      alert('Swap transaction failed. See console for details.');
    }
  };
```

Finally, attach this new handler to the Swap button:

```
// in the render, near the Simulate button
<button
  className="flex-1 bg-indigo-600 hover:bg-indigo-700 text-white font-
medium py-3 px-4 rounded-lg transition-all"
  onClick={handleSwap}
>
  Swap
</button>
```

Everything else stays the same as in Step #5.

---

7. Testing Your Swap

Now that your app is running, you can test the swap functionality:

1. **Connect your wallet** by clicking the "Connect Wallet" button and selecting your TON wallet from the modal.
2. **Select tokens** from the dropdown menus:
   - Choose a token you own in the "From" dropdown
   - Select a token you want to receive in the "To" dropdown
3. **Enter an amount** you wish to swap (make sure it's an amount you actually have in your wallet).
4. **Simulate the swap** by clicking the "Simulate" button to see the expected output amount before committing to the transaction.
5. **Execute the swap** by clicking the "Swap" button. This will prompt your wallet to approve the transaction.
6. **Confirm in your wallet** when the approval request appears.

Upon successful completion, the transaction will be processed on-chain, and your wallet balances will update accordingly. The whole process typically takes just a few seconds to complete.

---

8. Conclusion

You now have a React + Vite app with Tailwind CSS that:

- Connects to a TON wallet using TonConnect.
- Fetches available tokens from STON.fi.
- Simulates swaps (via simulateSwap).
- Builds and sends swap transactions (via @ston-fi/sdk).

Feel free to expand this demo with:

- Improved decimal handling for each token.
- Custom slippage tolerance settings.
- More robust error/success messages.

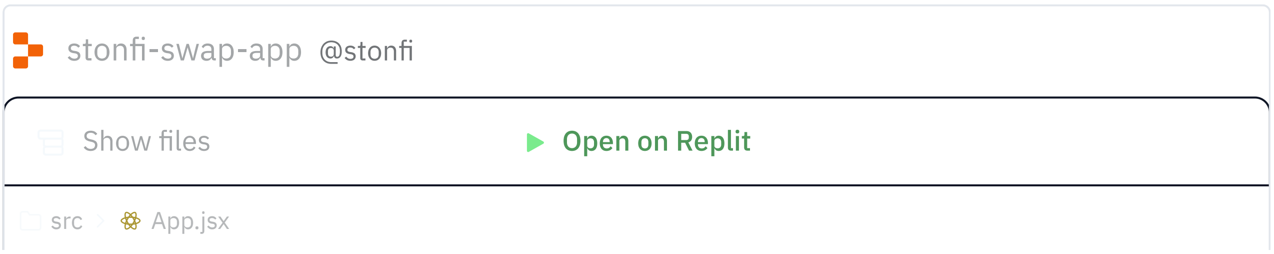
Happy building on TON and STON.fi!

---

9. Live Demo

With this Replit demo, you can:

- Open the project directly in your browser
- Fork the Replit to make your own copy
- Run the application to see it in action
- Explore and modify the code to learn how it works
- Experiment with different features and UI changes



```
import { useEffect, useState } from 'react';
import { dexFactory, Client } from "@ston-fi/sdk";
import { TonConnectButton, useTonConnectUI, useTonAddress } from '@tonconnect/ui-react';
import { StonApiClient, AssetTag } from '@ston-fi/api';

function App() {
  const [tonConnectUI] = useTonConnectUI();
  const userAddress = useTonAddress();
  const [assets, setAssets] = useState([]);
  const [fromAsset, setFromAsset] = useState(null);
  const [toAsset, setToAsset] = useState(null);
  const [amount, setAmount] = useState('');
  const [simulationResult, setSimulationResult] = useState(null);

  // Single function to handle changes in "From", "To", and "Amount"
  // Clears the simulation result each time any input changes
  const handleChange = (setter) => (e) => {
    const value = e.target.value;

    if (setter === setFromAsset || setter === setToAsset) {
```

10. Advanced Example App

For those seeking a feature-rich, more advanced approach, we also have a Next.js Demo App that:

- Uses Next.js for a scalable framework
- Utilizes hooks and providers for an elegant architecture
- Demonstrates better error handling, robust state management, and additional STON.fi features

You can explore the code in our repository:

[STON.fi SDK Next.js Demo App ↗]

Or see it in action at our live demo:

SDK Demo App ↗