**Adding Connect Functionality**

**Connecting Your Dapp: Adding a web3 Wallet Button with JavaScript and Viem**

This lesson guides you through adding a "Connect Wallet" button to a basic HTML page. We'll start with fundamental JavaScript DOM manipulation and event handling, then check for a browser wallet like MetaMask, and finally use the modern viem library to robustly handle the connection process.

**Initial Setup and Selecting the Button**

First, ensure you have a basic HTML file (index.html) and a JavaScript file (index-js.js) linked together. Your HTML needs a button element we can interact with.

index.html**:**

<!DOCTYPE html>

<html>

<head>

<title>Connect Button Lesson</title>

</head>

<body>

<button id="connectButton">Connect</button>

<script src="index-js.js" type="module"></script> <!-- type="module" is needed for imports -->

</body>

</html>

Now, in your index-js.js file, we need to get a reference to this button using JavaScript's Document Object Model (DOM) API. The document object represents your HTML page, and its methods allow us to find and manipulate elements.

index-js.js**:**

const connectButton = document.getElementById('connectButton');

This code uses document.getElementById to find the HTML element with the ID "connectButton" and stores a reference to it in the constant connectButton. We can now use this variable to control the button's behavior.

**Adding Basic Click Functionality**

To make the button interactive, we attach an event listener. Specifically, we'll define a function to run whenever the button is clicked.

index-js.js**:**

// Get the button element (from previous step)

const connectButton = document.getElementById('connectButton');

​

// Define the function to run on click

function connect() {

console.log("Button clicked!"); // Simple log to test

}

​

// Assign the function to the button's onclick event

connectButton.onclick = connect;

Here, we define a function named connect. For now, it just logs a message to the browser's developer console. We then assign this connect function to the onclick property of our connectButton element. Now, every time the button is clicked in the browser, the connect function will execute, and you'll see "Button clicked!" in the console.

**Checking for MetaMask (or Similar Wallets)**

Before attempting to connect a wallet, we must check if one is available in the user's browser environment. Ethereum wallets like MetaMask inject a global JavaScript object, typically window.ethereum, into the browser. We can check for its existence.

Let's modify the connect function:

index-js.js**:**

const connectButton = document.getElementById('connectButton');

function connect() {

// Check if window.ethereum is present

if (typeof window.ethereum !== "undefined") {

// Wallet is likely installed

console.log("MetaMask (or compatible wallet) is available!");

// We'll add connection logic here later

} else {

// Wallet is not installed

console.log("No wallet detected.");

connectButton.innerHTML = "Please install MetaMask!"; // Update button text

}

}

​

connectButton.onclick = connect;

Inside connect, we use typeof window.ethereum !== "undefined" to see if the ethereum object exists on the window. If it does, we log a success message (for now). If it doesn't, we log a different message and update the button's text (innerHTML) to prompt the user to install a wallet. Test this in browsers with and without MetaMask installed to see the difference.

**Introducing Wallet Abstraction with Viem**

While directly using window.ethereum works for basic checks and connections with MetaMask-compatible wallets, it's often better to use a wallet abstraction library. These libraries provide a standardized way to interact with various types of wallets and blockchain functionalities, making your code more robust, maintainable, and compatible with different wallet providers.

Two popular choices are ethers.js and viem. We'll use viem (viem.sh) for this tutorial due to its modern design, type safety, and lightweight nature.

Instead of installing viem via npm (common in larger projects), we'll import it directly from a Content Delivery Network (CDN) using an ES Module import, which works well for simple HTML/JS setups. Make sure your <script> tag in index.html has type="module".

index-js.js**:**

// Import necessary functions from viem via CDN

import { createWalletClient, custom } from "https://esm.sh/viem";

​

const connectButton = document.getElementById('connectButton');

​

// Declare walletClient variable outside the function

// so it can be potentially accessed elsewhere if needed

let walletClient;

​

function connect() {

if (typeof window.ethereum !== "undefined") {

console.log("MetaMask (or compatible wallet) is available!");

// Connection logic using viem will go here

} else {

console.log("No wallet detected.");

connectButton.innerHTML = "Please install MetaMask!";

}

}

​

connectButton.onclick = connect;

We import createWalletClient (to create an interface for wallet actions) and custom (to specify how to communicate with the wallet) from viem.

**Connecting the Wallet using Viem**

Now, let's use viem inside our if block to handle the actual connection request.

In viem, we create a WalletClient. This client needs a "Transport" to define *how* it communicates. Since we want to use the browser's injected wallet (MetaMask), we use the custom transport configured with window.ethereum.

Once we have the client, we can call its requestAddresses method. This method corresponds to the standard eth\_requestAccounts RPC call, which prompts the user to connect their wallet.

index-js.js**(inside the**connect**function's**if**block):**

// Inside the 'if (typeof window.ethereum !== "undefined")' block:

console.log("Connecting using viem...");

​

// Create a Wallet Client

walletClient = createWalletClient({

transport: custom(window.ethereum) // Use the browser's injected provider

});

​

// Request wallet connection (account addresses)

walletClient.requestAddresses();

​

console.log("Connection request sent..."); // This logs too early! (See next section)

connectButton.innerHTML = "Connected!"; // This updates too early!

If you run this code and click connect, MetaMask *will* pop up asking for permission. However, you'll notice the console logs "Connection request sent..." and the button text changes to "Connected!" *immediately*, even before you interact with the MetaMask pop-up. This happens because requestAddresses is an asynchronous operation.

**Handling Asynchronicity with**async**/**await

Wallet interactions and network requests are asynchronous: they don't block the rest of your code while waiting for a response (like user confirmation in MetaMask). They return a Promise, which represents a future result.

To make our code wait for the asynchronous operation to complete before proceeding, we use the async and await keywords.

1. Mark the function containing the asynchronous call as async.
2. Use the await keyword before the asynchronous call (walletClient.requestAddresses()).

index-js.js**(Refactored**connect**function):**

import { createWalletClient, custom } from "https://esm.sh/viem";

​

const connectButton = document.getElementById('connectButton');

let walletClient;

​

// Make the function async

async function connect() {

if (typeof window.ethereum !== "undefined") {

console.log("Connecting...");

​

walletClient = createWalletClient({

transport: custom(window.ethereum),

});

​

try {

// Wait for the user to connect their wallet

const addresses = await walletClient.requestAddresses();

console.log("Connected accounts:", addresses); // Log the connected address(es)

​

// This code now runs ONLY AFTER the await completes successfully

connectButton.innerHTML = `Connected: ${addresses[0].slice(0, 6)}...`; // Show part of address

console.log("Connection successful!");

} catch (error) {

// Handle errors, like the user rejecting the connection

console.error("Connection failed:", error);

connectButton.innerHTML = "Connect"; // Reset button text on failure

}

​

} else {

connectButton.innerHTML = "Please install MetaMask!";

}

}

​

connectButton.onclick = connect;

Now, when connect is called:

1. It checks for window.ethereum.
2. It creates the walletClient.
3. It hits await walletClient.requestAddresses(). The function execution *pauses* here.
4. MetaMask (or the relevant wallet) prompts the user.
5. If the user approves, the Promise returned by requestAddresses resolves, await gets the result (an array of addresses), and execution continues. The button text is updated, and success is logged.
6. If the user rejects or an error occurs, the catch block executes, logging the error and resetting the button text.

**Important Considerations**

* **Brave Shields:** If using the Brave browser, ensure "Shields" are down for your development environment (e.g., localhost or 127.0.0.1) as they can sometimes interfere with wallet detection and interaction.
* **Asynchronicity is Crucial:** Understanding async/await and Promises is fundamental in web3, as nearly all interactions with wallets and blockchains are asynchronous.
* **Error Handling:** The try...catch block is essential for a good user experience. Always handle potential errors, such as the user cancelling the connection request.
* **Viem Benefits:** Using viem provides type safety, better abstraction, and helper functions compared to directly using window.ethereum, making development smoother, especially for more complex applications.

You have now successfully implemented a "Connect Wallet" button that uses modern JavaScript and the viem library to interact with browser-based Ethereum wallets.