**Connecting To Different Chains**

**Connecting Your Web App to MetaMask**

Connecting a decentralized application (dApp) to a user's crypto wallet is the foundational step for enabling blockchain interactions. It allows your web application to request information like the user's wallet address and to prompt the user to sign transactions or messages. This lesson demonstrates the basic process of connecting a web app to the MetaMask browser extension wallet using JavaScript and the viem library.

**The Connection Workflow**

The typical flow involves the user initiating a connection from the dApp's interface, which then triggers the wallet extension to ask for the user's permission.

1. **Initiation:** The user clicks a "Connect" button within the web application.
2. **Wallet Prompt:** If the site isn't already connected or authorized, MetaMask (or the relevant wallet) pops up, asking the user to confirm they want to connect the site. This prompt typically shows the site's URL and the permissions being requested (e.g., "See your accounts").
3. **User Confirmation:** The user approves the connection request within the MetaMask interface.
4. **Application Update:** The web application receives confirmation of the connection, usually obtaining the user's wallet address(es). The UI is then updated to reflect the connected state (e.g., changing the button text to "Connected!" or displaying the user's address).

**Managing Connections and Permissions in MetaMask**

Once a site is connected, users retain full control over the connection via the MetaMask extension interface:

* **Viewing Connections:** Users can see which sites are connected to which accounts.
* **Permissions Management:** Within MetaMask, by navigating to the settings for a specific connected site (often via a three-dot menu next to the connected indicator), users can view and edit the permissions granted. This includes:
  + **Account Access:** Controlling which specific accounts the site can see.
  + **Network Access:** Controlling which blockchain networks the site can suggest transactions for or interact with.
* **Disconnecting:** Users can explicitly disconnect a site at any time through the MetaMask interface. When disconnected this way, the application's connection status is immediately revoked.

**Implementing a Basic Connection with**viem

Let's examine a simple JavaScript implementation using the viem library to handle the connection logic. This code assumes you have an HTML button with the ID connectButton.

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

​

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

​

let walletClient; // Variable to hold the wallet client instance

​

async function connect() {

// 1. Check if the MetaMask provider (window.ethereum) is available

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

console.log("MetaMask is installed!");

​

try {

// 2. Create a Wallet Client using viem's custom transport

// This configures viem to use MetaMask's injected provider

walletClient = createWalletClient({

transport: custom(window.ethereum),

});

​

// 3. Request access to the user's accounts

// This triggers the MetaMask connection prompt if not already authorized

await walletClient.requestAddresses();

​

// 4. Update the UI to indicate a successful connection

connectButton.innerHTML = "Connected!";

​

// Now you can use walletClient for further interactions

// e.g., const accounts = await walletClient.getAddresses();

// console.log("Connected accounts:", accounts);

​

} catch (error) {

// Handle potential errors during connection (e.g., user rejection)

console.error("Failed to connect:", error);

connectButton.innerHTML = "Connection Failed";

}

​

} else {

// 5. Update UI if MetaMask is not detected

connectButton.innerHTML = "Please install MetaMask!";

}

}

​

// 6. Attach the connect function to the button's click event

connectButton.onclick = connect;

​

**Code Breakdown:**

1. **Check for MetaMask:** It verifies if window.ethereum exists, which is the standard way MetaMask injects its provider API into the browser.
2. **Create Wallet Client:** viem's createWalletClient function is used to create an instance that allows interaction with the user's wallet. The custom(window.ethereum) transport tells viem to communicate through the MetaMask provider.
3. **Request Addresses:** walletClient.requestAddresses() is the key asynchronous call. It asks MetaMask for access to the user's accounts. If the site hasn't been connected before, or if permissions are needed, MetaMask will show its pop-up confirmation dialog to the user. If the site is already authorized, this might resolve immediately without a pop-up.
4. **Update UI on Success:** If requestAddresses() completes successfully (the user approves or was already connected), the button text is updated.
5. **Handle Missing MetaMask:** If window.ethereum is not found, the UI informs the user to install MetaMask.
6. **Event Listener:** The connect function is assigned to run when the connectButton is clicked.

**The Challenge of Connection State Persistence**

A crucial aspect not covered by the simple code above is managing the connection state across page refreshes.

* **Explicit Disconnect:** If the user explicitly disconnects the site via the MetaMask interface and then refreshes the page, the application correctly shows the "Connect" state.
* **Refresh While Connected:** However, if the user connects, *then* simply refreshes the page, the simple code example will revert to showing "Connect". Clicking "Connect" again might result in an *instantaneous* UI update to "Connected!" without a MetaMask prompt. This happens because MetaMask remembers that the site was previously authorized.

Handling this "remembered" state robustly requires additional logic in the web application. The application needs to check on page load if it has a pre-existing authorization with the wallet and automatically re-establish the connected state if possible. This involves managing flags, potentially checking connection status silently, and updating the UI accordingly.

Implementing this logic correctly can significantly increase the complexity of the connection code. In real-world dApps, this state management, along with handling different wallets and connection edge cases, is often delegated to specialized libraries (like Web3Modal, RainbowKit, ConnectKit, etc.) which abstract away this complexity.

For now, understanding the fundamental requestAddresses flow and the user's control via MetaMask provides a solid foundation for enabling blockchain interactions in your web applications.