Web3 Solana blockchain - 04

What is a provider?

Node providers are essentially teams/individuals/organizations that offer a way to access the information on a blockchain without having to run your own node!

Node: Running on a physical computer software.connects with the rest of the blockchain network. And A Blockchain network is made up of nodes

Node Provide: Nodes are hard to setup, manage etc. So Node providers offer way to access information on a blockchain without having to run your own node.

Phantom wallet provides us with the connection to connect to the solana blockchain

Use React To build a Solana based dapp

```
## Phantom Wallet Connect

const connect Wallet = async () => {

// @ts-ignore
   const { solana } = window;
// checks if phantom wallet exists
    if (solana) {
       try {
// connects wallet and returns response which includes the wallet public key

          const response = await solana.connect();
          console.log('wallet account ', response.publicKey.toString());
          // update walletKey to be the public key
          setWalletKey(response.publicKey.toString());
} catch (err) {
console.log(err);
}
}
};
```

1. `const { solana } = window;`
   - This line uses destructuring assignment to extract the `solana` object from the `window` object. It assumes that the `solana` object is available in the global scope, possibly provided by a wallet extension or library.
2. `if (solana) { ... }`
   - This condition checks if the `solana` object exists. It verifies if the wallet integration is available, as expected.
3. `try { ... } catch (err) { ... }`
   - This block sets up a try-catch statement to handle any potential errors that might occur during the execution of the code within the try block.
4. `const response = await solana.connect();`
   - This line calls the `connect()` method on the `solana` object, which is assumed to be a function provided by the wallet integration. It establishes a connection with the wallet and returns a response object.
5. `console.log('wallet account ', response.publicKey.toString());`
   - This line logs the wallet account or public key, obtained from the `response` object, to the console. The `response.publicKey` is assumed to be a property that holds the public key.
6. `setWalletKey(response.publicKey.toString());`
   - This line assumes that there is a `setWalletKey` function available, which is used to update the `walletKey` state variable. It updates `walletKey` with the value of `response.publicKey.toString()`, which converts the public key to a string representation.
7. `catch (err) { console.log(err); }`
   - This block catches any errors that occur within the try block and logs them to the console. It provides a basic error handling mechanism.

```
## Phantom Wallet Disconnect


  const disconnectWallet = async () => {
    // @ts-ignore
    const { solana } = window;


                // checks if phantom wallet exists


    if (solana) {


      try {
                                  // DISCONNECT Wallet
        const response = await solana.disconnect();


        // Reset Wallet PublicKey


        setWalletKey(undefined);


        console.log('Wallet Disconnect')
        } catch (err) {
          console.log(err);
        }
      }
    };
```

1. `const { solana } = window;`
   - This line uses destructuring assignment to extract the `solana` object from the `window` object. It assumes that the `solana` object is available in the global scope, possibly provided by a wallet extension or library.
2. `if (solana) { ... }`
   - This condition checks if the `solana` object exists. It verifies if the wallet integration is available, as expected.
3. `try { ... } catch (err) { ... }`
   - This block sets up a try-catch statement to handle any potential errors that might occur during the execution of the code within the try block.
4. `const response = await solana.disconnect();`
   - This line calls the `disconnect()` method on the `solana` object, which is assumed to be a function provided by the wallet integration. It initiates the disconnection process and returns a response object.
5. `setWalletKey(undefined);`
   - This line assumes that there is a `setWalletKey` function available, which is used to update the `walletKey` state variable. It sets the `walletKey` to `undefined`, effectively resetting it.
6. `console.log('Wallet Disconnect');`
   - This line logs a message to the console indicating that the wallet has been disconnected.
7. `catch (err) { console.log(err); }`
   - This block catches any errors that occur within the try block and logs them to the console. It provides a basic error handling mechanism.