**[NOTES] fetching data**

1. const fetchMessage = async () => {
2. const response = await fetch('/api/hello'); // This gets the response object (headers + body)
4. // Now you have the response object, but you still need to read the body
5. const text = await response.text(); // This reads the body, which may still be streaming
6. console.log(text); // Logs the full text response
7. };

Why Does response.text() Need await?

Even after you've used await fetch(), which waits for the response from the server, the response.text() method still returns a promise. The reason for this is that the **body of the response might not be fully available yet** when you first get the response object.

When the server sends data back, it may send it in pieces (especially if it's a larger file or streamed content), and JavaScript needs to make sure the entire response body is received before converting it to text. This is why response.text() also uses await—it’s waiting for the full response content to arrive.

A Quick Breakdown:

1. **await fetch()**: This waits for the server to respond, but it only ensures that the basic response info (like headers and status) is received.
2. **await response.text()**: This waits for the body (the actual content) to be fully received and then converts it to text.

Even if the response is small and quick, JavaScript handles all this asynchronously to avoid blocking the rest of the page while it's waiting for the data.

In short, await ensures that JavaScript waits until everything is ready, making it safe to use the text content without running into issues like incomplete data.