JavaScript `async/await` and Fetch API

1. `async` and `await`

`async` and `await` are keywords in JavaScript that make working with **Promises** easier and more readable. They help you write asynchronous code (code that doesn't block other operations) that looks like regular, synchronous code.

- **`async`** : Declares a function as **asynchronous**, meaning it will return a Promise.
- **`await`** : Pauses the function execution until the Promise is resolved (successful) or rejected (failed), making the code wait for the result.

Example:

```
async function fetchData() {
  let result = await Promise.resolve("Data loaded");
  console.log(result); // Output: Data loaded
}
fetchData();
```

In this example:

- The `async` function `fetchData` contains `await`, which waits for the Promise to resolve before logging the result.

---

2. **Fetch API**

The **Fetch API** is used to make HTTP requests (like getting data from a server) in JavaScript. It returns a **Promise** that resolves when the data is fetched or rejects if there's an error.

Example of Fetch API:

```
fetch('https://api.example.com/data')
  .then(response => response.json()) // Converts the response to JSON
  .then(data => console.log(data)) // Logs the data
  .catch(error => console.error('Error:', error)); // Handles errors
```

This code:

- Uses `fetch()` to make a GET request to the given URL.
- When the data is received, it's converted to JSON using `.json()`.
- If successful, the data is logged. If it fails, the `catch()` handles the error.

---

3. `async` / `await` with Fetch API

Using `async` / `await` makes it easier to work with the **Fetch API** without the need for chaining `.then()` calls.

Example of `async` / `await` with Fetch API:

```
async function getData() {
  try {
    let response = await fetch('https://api.example.com/data');
    let data = await response.json(); // Waits for the response to be converted to JSON
    console.log(data); // Logs the data
  } catch (error) {
    console.error('Error:', error); // Handles any errors
  }
}

getData();
```

In this example:

- `async` declares `getData` as an asynchronous function.

- `await` pauses the function execution until the `fetch` request and `.json()` conversion complete.
- `try...catch` is used to handle errors (like network issues).

In Simple Terms:

- **`async/await`** : Makes asynchronous code (like waiting for data) look like normal, step-by-step code.
- **Fetch API**: A way to request data from a server. It returns a Promise and handles responses from web services.
- Combining `async/await` with Fetch makes handling server requests easier and cleaner than using `.then()` chains.