Perfect. Continuing **systematically**, **NotebookLM-ready**, with **NOTES and CODE strictly separated**.

📘 TOPIC: Fetch API

(HTTP ● Promises ● `async/await` ● Errors ● Internals ● Interview Traps)

🟦 NOTE 1: What `fetch` really is

- `fetch` is **NOT JavaScript language**
- It is a **Web API** (Browser + modern Node)
- Returns a **Promise**
- Designed for **network requests (HTTP)**

👉 `fetch` does **not** reject on HTTP errors (4xx / 5xx)

🟦 NOTE 2: Basic fetch lifecycle (internal flow)

1. JS calls `fetch()`
2. Request delegated to runtime (browser / Node)
3. Network happens outside JS engine
4. Response comes back
5. Promise is resolved
6. Microtask queue schedules `.then` / `await`

🟩 CODE 1: Basic fetch request

```
js

fetch("https://api.example.com/data")
  .then(res => res.text())
  .then(data => console.log(data));
```

🟦 NOTE 3: Fetch always returns a Promise

- Promise resolves to a **Response object**
- Response contains:
  - status
  - headers
  - body (stream)

🟩 CODE 2: Response object inspection

```
js

fetch(url).then(response => {
  console.log(response.status);
```

```
  console.log(response.ok);
});
```

🟦 NOTE 4: `response.ok` (IMPORTANT)

- `true` → status 200–299
- `false` → 4xx / 5xx
- Promise is STILL resolved even if `ok === false`

🟩 CODE 3: HTTP error trap

```
js

fetch("/404")
  .then(res => {
    console.log(res.ok); // false
  });
```

🟦 NOTE 5: Parsing response body

Body can be read **only once**.

Common methods:

- `.json()`
- `.text()`
- `.blob()`
- `.arrayBuffer()`

🟩 CODE 4: Parsing JSON

```
js

fetch("/api/user")
  .then(res => res.json())
  .then(data => console.log(data));
```

🟦 NOTE 6: Why `.json()` is async

- Body is a **stream**
- Parsing happens asynchronously
- `.json()` returns a Promise

🟩 CODE 5: Body used once

```
js
```

```
fetch(url).then(res => {
  res.json();
  // res.text(); ❌ error: body already used
});
```

🟦 NOTE 7: Using `async/await` with fetch (recommended)

- Cleaner
- Easier error handling
- Linear flow

🟩 CODE 6: Fetch with async/await

```
js

async function loadData() {
  const res = await fetch("/api/data");
  const data = await res.json();
  console.log(data);
}

loadData();
```

🟦 NOTE 8: Handling HTTP errors correctly

You must **manually check** `response.ok`.

🟩 CODE 7: Proper error handling

```
js

async function load() {
  const res = await fetch("/api/data");

  if (!res.ok) {
    throw new Error(`HTTP Error ${res.status}`);
  }

  const data = await res.json();
  return data;
}
```

🟦 NOTE 9: Network errors vs HTTP errors

Type	Promise
Network error	❌ rejected
404 / 500	✅ resolved

🟩 CODE 8: Network error example

```
js

fetch("https://invalid.domain")
  .catch(err => console.log("Network error"));
```

🟦 NOTE 10: Sending data (POST request)

- Use `method`
- Set `headers`
- Send `body`

🟩 CODE 9: POST request

```
js

fetch("/api/user", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ name: "Anoop" })
});
```

🟦 NOTE 11: Fetch options object

Common options:

- method
- headers
- body
- mode
- credentials
- cache

🟩 CODE 10: Fetch options

```
js
```

```
fetch(url, {
  method: "GET",
  headers: { "Accept": "application/json" }
});
```

🟦 NOTE 12: CORS (very important)

- Browser security feature
- Server decides access
- Node.js does NOT enforce CORS

🟩 CODE 11: CORS error (browser)

```
js

fetch("https://other-site.com/api");
// ❌ blocked by browser if CORS headers missing
```

🟦 NOTE 13: AbortController (cancel fetch)

- Used to cancel in-flight requests
- Important for performance (React, SPA)

🟩 CODE 12: Abort fetch

```
js

const controller = new AbortController();

fetch(url, { signal: controller.signal });

controller.abort();
```

🟦 NOTE 14: Parallel fetch requests

- Use `Promise.all`
- Improves performance

🟩 CODE 13: Parallel fetch

```
js

const [a, b] = await Promise.all([
  fetch("/api/a").then(r => r.json()),
  fetch("/api/b").then(r => r.json())
]);
```

🟦 NOTE 15: Common interview traps

- Fetch doesn’t reject on 404
- `.json()` returns a Promise
- Body can be consumed once
- CORS is browser-only
- Fetch is async even for local files

🟦 FINAL MENTAL MODEL

- `fetch` = Web API
- Returns Promise → Response
- HTTP errors ≠ Promise rejection
- Must check `response.ok`
- Body is a stream
- Use `async/await` + try/catch
- Handle CORS and cancellation

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ Error Handling (sync vs async, try/catch pitfalls)
2️⃣ Memory Management & Garbage Collection
3️⃣ Browser Storage (localStorage, sessionStorage)

Reply with **next** or the **number** you want to continue.