`Fetch()` And `safeFetch()`

☑ 1. What does `fetch()` return?

- The `fetch()` function **always returns a** `Promise`.
- This promise:
  - ☑ **resolves** when the **network request completes successfully** (regardless of status code).
  - ❌ **rejects** only for **network errors** (e.g., offline, DNS error, CORS error).

☑ 2. What happens on `Promise` **Resolve**?

When the `fetch` Promise **resolves**, it gives you a **Response object**:

```
fetch("https://api.example.com/data")
  .then(response => {
    console.log(response instanceof Response); // true
    console.log("Status Code:", response.status); // e.g., 200, 404, 500
    return response.json(); // assuming JSON response
  })
  .then(data => {
    console.log("Parsed Data:", data);
  });
```

🎯 Key Properties of `Response` Object:

Property	Type	Description
`status`	Number	HTTP status code (e.g., 200, 404)
`ok`	Boolean	`true` if status in 200–299
`statusText`	String	Human-readable status (e.g., OK, Not Found)
`headers`	Headers	Response headers
`json()`, `text()`	Function	Methods to read the body

> ◇ If the server returns `404 Not Found`, the fetch **still resolves**. But `response.ok` will be `false`.

❌ 3. What happens on `Promise` **Reject**?

The `fetch()` Promise **rejects** (goes to `.catch()`) only when:

- Network fails (DNS error, no internet)

- Request is blocked (CORS error)
- Invalid URL (e.g., `fetch("")`)
- Connection timeout (if manually configured)

```
fetch("https://invalid-domain.test")
  .then(res => res.json())
  .catch(err => {
    console.error("❌ Caught Error:", err);
    console.log(err.name);    // TypeError
    console.log(err.message); // Failed to fetch or NetworkError
  });
```

🔘 Common Error Object Info:

Property	Example
`err.name`	`"TypeError"`
`err.message`	`"Failed to fetch"` (browser-dependent)

🧪 Full Example: Handling All Cases

```
fetch("https://api.example.com/data")
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => {
    console.log("✅ Data received:", data);
  })
  .catch(error => {
    console.error("❌ Error occurred:", error);
    console.log("Name:", error.name);
    console.log("Message:", error.message);
  });
```

🧠 Summary Table:

Scenario	Does `fetch` resolve?	Is `response.ok` true?	Do you get `.catch()`?	Error type
200 OK	✅ Yes	✅ Yes	❌ No	—

Scenario	Does `fetch` resolve?	Is `response.ok` true?	Do you get `.catch()`?	Error type
404 Not Found	☑ Yes	❌ No	❌ No (unless thrown)	—
500 Internal Server Error	☑ Yes	❌ No	❌ No (unless thrown)	—
DNS error / Invalid domain	❌ No	—	☑ Yes	`TypeError`
CORS blocked	❌ No	—	☑ Yes	`TypeError`
No internet	❌ No	—	☑ Yes	`TypeError`

safeFetch()

- ☑ `response.ok` checks (status code validation)
- ❌ error catching (`try/catch`)
- 🧠 customizable error messages
- 🔁 optional retry logic (optional extension)

☑ SafeFetch: Basic Version

🔧 Code:

```
async function safeFetch(url, options = {}) {
  try {
    const response = await fetch(url, options);

    if (!response.ok) {
      // HTTP error (but not a network failure)
      throw new Error(`❌ HTTP Error: ${response.status}
${response.statusText}`);
    }

    const data = await response.json(); // or .text(), .blob(), etc.
    return {
      success: true,
      data,
      status: response.status,
    };

  } catch (error) {
    return {
      success: false,
      error: {
        name: error.name,
```

```
        message: error.message,
        isNetworkError: error instanceof TypeError,
      },
    };
  }
}
```

🧪 Example Usage:

```
(async () => {
  const result = await safeFetch("https://jsonplaceholder.typicode.com/posts/1");

  if (result.success) {
    console.log("✅ Data:", result.data);
  } else {
    console.error("❌ Error Details:", result.error);
    if (result.error.isNetworkError) {
      console.warn("🌐 Check your internet or CORS settings.");
    }
  }
})();
```

🧠 Returned Object Structure

✅ On Success:

```
{
  success: true,
  data: { /* parsed JSON or text */ },
  status: 200
}
```

❌ On Error:

```
{
  success: false,
  error: {
    name: "TypeError",
    message: "Failed to fetch",
    isNetworkError: true
  }
}
```

💡 Optional Advanced Version (With Retry)

```
async function safeFetchWithRetry(url, options = {}, retries = 3) {
  for (let attempt = 1; attempt <= retries; attempt++) {
    const result = await safeFetch(url, options);
    if (result.success) return result;
    console.warn(`🔁 Retry ${attempt} failed. Retrying...`);
  }
  return {
    success: false,
    error: {
      message: `Failed after ${retries} retries.`,
    },
  };
}
```

🔍 Summary Table

Feature	Native Fetch	`safeFetch()`
Returns parsed data	❌ (manual)	☑
Handles non-200 status codes	❌	☑
Handles network errors	❌	☑
Easy success/error flags	❌	☑
Optional retry mechanism	❌	☑ (advanced)

📦 Final Thoughts

☑ `safeFetch()` improves **developer experience** and reduces repetitive boilerplate in API-heavy apps.

It looks like your code has a few syntax and logical issues — no worries! Here's the **corrected and clean version** of the code you meant to write, along with clear inline comments and emoji-enhanced logging:

☑ Fetch With Real Example

```
// 🌐 API Endpoint
const API_URL = "https://api.github.com/users/akshaymarch7";

// ☑ Async function to handle the fetch promise
async function handlePromise() {
  try {
    console.log("🚀 Fetching data from GitHub API...");
```

```
    // 👇 Await the response
    const response = await fetch(API_URL);

    // ✅ Check if response is OK (status 200-299)
    if (!response.ok) {
      throw new Error(`❌ HTTP Error: ${response.status}`);
    }

    // 📦 Parse response JSON
    const data = await response.json();

    // 📋 Log the final JSON data
    console.log("✅ Parsed JSON Value:");
    console.log(data);
  } catch (error) {
    console.error("❌ Error while fetching or parsing:");
    console.error(error);
  }
}

// ▶️ Run the function
handlePromise();
```

📦 Output (Example)

```
🚀 Fetching data from GitHub API...
✅ Parsed JSON Value:
{
  login: "akshaymarch7",
  id: 123456,
  avatar_url: "https://...",
  ... // more user data
}
```

✅ `fetch()` with Arrow Functions Inside `handlePromise`

```
const API_URL = "https://api.github.com/users/akshaymarch7";

async function handlePromise() {
  console.log("🚀 Initiating fetch request...");

  fetch(API_URL)
    .then((response) => {
      if (!response.ok) {
        throw new Error(`❌ HTTP error! Status: ${response.status}`);
      }
```

```
      return response.json(); // ⬅️ returns a promise
    })
    .then((jsonValue) => {
      console.log(jsonValue);
    })
    .catch((error) => {
      console.error("❌ Error occurred during fetch or parsing:");
      console.error(error);
    });
}

handlePromise();
```

🧠 How It Works

Phase	Explanation
`fetch()`	Starts an HTTP request and returns a `Promise<Response>`
`.then(res => …)`	Handles the successful HTTP response
`res.json()`	Converts the response to JSON (also returns a promise)
`.then(data => …)`	Handles the parsed JSON object
`.catch(err => …)`	Catches network or parsing errors

```
fetch(API_URL)
  .then(response => response.json())
  .then(jsonValue => console.log(jsonValue))
  .catch(error => console.error(error));
```
