Understanding `fetch()` API in JavaScript

The `fetch()` API is a modern way to make HTTP requests in JavaScript. It returns a Promise and is used to interact with APIs over the network.

Why Do We Need fetch in JavaScript?

`fetch()` lets JavaScript communicate with servers — to **get, send, update, or delete** data — without reloading the entire web page.

Most modern web applications need `fetch()` to:

- Get user data (e.g., profile info)
- Submit forms (e.g., login, comments)
- Display dynamic content (e.g., tweets, product prices)
- Work with external services (e.g., weather, maps, payments)

We will understand the working of `fetch` for 4 different scenarios:

1. fetch on a simple static URL
2. fetch on a dynamic URL
3. fetch with data/info sent as request header (e.g token)
4. fetch with data/info sent as request body

Syntax:

```
fetch(url, options)
  .then(response => response.json())
  .then(data => console.log('Data: ', data))
  .catch(error => console.error('Error: ', error));
```

1. Fetch on a Simple Static URL

Use case: Get a list of users from a public API.

Imagine you want to retrieve a list of active users of an application — you don't need to give any information; you just request and get a list.

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json())
  .then(data => console.log('Users:', data))
  .catch(error => console.error('Error: ', error));
```

2. Fetch on a Dynamic URL

Use case: Get a specific user's data using their `userId`.

You want to retrieve information for a specific user with the help of their `userId`.

```
const userId = 3;
fetch(`https://jsonplaceholder.typicode.com/users/${userId}`)
  .then(response => response.json())
  .then(data => console.log(`User ${userId}:`, data))
  .catch(error => console.error('Error:', error));
```

3. Fetch with Data Sent as Request Header (e.g., Token)

Use case: You need to send data/info in the request headers.

You want to access a protected route which needs a token to be sent along with its request header.

```
const token = 'fake-jwt-token-123';

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${token}`
  }
})
  .then(response => response.json())
  .then(data => console.log('Protected Data:', data))
  .catch(error => console.error('Error:', error));
```

4. Fetch with Data Sent in the Request Body (POST/PUT)

Use case: Send data in the form of request body.

Suppose you are interacting with a blog website and need to create a new blog. The blog data must then be sent as request body.

```
const postData = {
  title: 'My First Post',
  body: 'This is the content of the post.',
  userId: 1
};

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(postData)
})
  .then(response => response.json())
  .then(data => console.log('Created Post:', data))
  .catch(error => console.error('Error:', error));
```

When Does a `fetch()` Request Fail?

You might expect `fetch()` to treat all non-200 responses (like 404 or 500) as errors — but that's not how it works.

`fetch()` API only rejects the Promise on network-level errors, not on HTTP status codes.

What is considered as an Error?

`fetch()` will throw an error (reject the Promise) for:

- Network failure (e.g., no internet connection)
- Invalid domain (e.g., server doesn't exist)
- CORS violations (when server blocks cross-origin requests)
- Request timeout (only if implemented manually or via `AbortController`)

```
fetch('https://this-does-not-exist.xyz/api') // invalid domain
  .then(response => response.json())  // This is never reached
```

```
    .catch(error => console.error('Network Error:', error));
```

What is NOT considered as an Error?

- 404 Not Found
- 403 Forbidden
- 500 Internal Server Error

HTTP errors are considered successful fetches, however, they must be handled manually in the code.

```
fetch('https://existing-domain/unknown-endpoint') //throws Not Found Error
  .then(response => {
    if (!response.ok) {
      throw new Error(`HTTP Error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => console.error('Handled Error:', error));
```

Summarizing the error handling in `fetch()`

Scenario	`fetch()` rejects the Promise automatically?	To be Handled manually?
No internet / DNS fail	Yes	No. (Rejected)
404 Not Found	No	Yes (`response.ok`)
500 Internal Server Error	No	Yes
401 Unauthorized	No	Yes
CORS blocked	Yes	No. (Rejected)