Fetch API and HTTP requests

🌐 Fetch API and HTTP Requests in JavaScript — Explained in Detail

📌 What is the Fetch API?

The **Fetch API** is a modern JavaScript interface for making **HTTP requests**. It replaces the older `XMLHttpRequest` and is **promise-based**, making it easy to use with `async/await`.

```
fetch(url, options) // Returns a Promise
```

✅ Basic Syntax

```
fetch("https://api.example.com/data")
  .then(response ⇒ response.json()) // Converts the response to JSON
  .then(data ⇒ console.log(data))
  .catch(error ⇒ console.error("Error:", error));
```

Or using `async/await`:

```
async function getData() {
  try {
    const response = await fetch("https://api.example.com/data");
    const data = await response.json();
    console.log(data);
  } catch (err) {
    console.error("Fetch error:", err);
  }
}
```

📦 Response Object

The `response` object returned from `fetch()` has important methods:

- `response.ok` → `true` if status code is 200–299
- `response.status` → HTTP status code
- `response.json()` → parses response body to JSON
- `response.text()` → returns raw text
- `response.blob()` → for binary data (e.g. images, files)

🧾 Making POST Requests

```
fetch("https://api.example.com/posts", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ title: "Hello", body: "World" })
})
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error("Error:", err));
```

🔐 Sending Authenticated Requests

```
fetch("https://api.example.com/user", {
  headers: {
    Authorization: "Bearer your_token_here"
  }
})
  .then(res => res.json())
  .then(data => console.log(data));
```

⚠️ Handling Errors Properly

The `fetch()` **only rejects** the promise for **network errors**, not for HTTP errors (like 404 or 500). You must check the response:

```
async function safeFetch() {
  const res = await fetch("https://api.example.com/data");
  if (!res.ok) {
    throw new Error(`HTTP error! Status: ${res.status}`);
  }
  const data = await res.json();
  return data;
}
```

📂 Common HTTP Methods

Method	Purpose
GET	Read/fetch data
POST	Create new data
PUT	Replace existing data
PATCH	Update part of the data
DELETE	Remove data

🔁 Example: PUT Request

```
fetch("https://api.example.com/posts/1", {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ title: "Updated Title" })
})
  .then(res ⇒ res.json())
  .then(data ⇒ console.log(data));
```

🌍 Cross-Origin Requests (CORS)

If you make requests to another domain (not your own), the browser enforces **CORS (Cross-Origin Resource Sharing)**. The server must set proper headers like:

```
Access-Control-Allow-Origin: *
```

Or a specific origin:

```
Access-Control-Allow-Origin: https://yourdomain.com
```

📥 Downloading Files

```
fetch("image.jpg")
  .then(response => response.blob())
  .then(blob => {
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "image.jpg";
    a.click();
  });
```

🧠 Summary

- The **Fetch API** is cleaner, promise-based, and modern.
- It's flexible: supports all HTTP methods, custom headers, JSON body.
- Always check `response.ok` for HTTP-level errors.
- Combine with `async/await` for readable, maintainable code.

Let me know if you'd like a hands-on project example using Fetch (e.g., a weather app, GitHub API, or form submission).