The **fetch() method** is used to send the requests to the server without refreshing the page. It is an alternative to the XMLHttpRequest object.

JavaScript can send network requests to the server and load new information whenever it’s needed.

For example, we can use a network request to:

* Submit an order,
* Load user information,
* Receive latest updates from the server,
* …etc.

…And all of that without reloading the page!

There’s an umbrella term “AJAX” (abbreviated **A**synchronous **J**avaScript **A**nd **X**ML) for network requests from JavaScript. We don’t have to use XML though: the term comes from old times, that’s why that word is there. You may have heard that term already.

The basic syntax of a fetch() request is as follows:

|  |
| --- |
| fetch(url, {options})  .then(data => {      // Do some stuff here  })  .catch(err => {      // Catch and display errors  }) |

The difference between XMLHttpRequest and fetch is that fetch uses [Promises](https://www.geeksforgeeks.org/javascript-promises/) which are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

The basic syntax is:

let promise = fetch(url, [options])

* **url** – the URL to access.
* **options** – optional parameters: method, headers etc.

Without options, that is a simple GET request, downloading the contents of the url.

The browser starts the request right away and returns a promise that the calling code should use to get the result.

Getting a response is usually a two-stage process.

**First, the promise, returned by fetch, resolves with an object of the built-in**[**Response**](https://fetch.spec.whatwg.org/#response-class)**class as soon as the server responds with headers.**

At this stage we can check HTTP status, to see whether it is successful or not, check headers, but don’t have the body yet.

The promise rejects if the fetch was unable to make HTTP-request, e.g. network problems, or there’s no such site. Abnormal HTTP-statuses, such as 404 or 500 do not cause an error.

We can see HTTP-status in response properties:

* **status** – HTTP status code, e.g. 200.
* **ok** – boolean, true if the HTTP status code is 200-299.

**Second, to get the response body, we need to use an additional method call.**

Response provides multiple promise-based methods to access the body in various formats:

* **response.text()** – read the response and return as text,
* **response.json()** – parse the response as JSON,
* **response.formData()** – return the response as FormData object (explained in the [next chapter](https://javascript.info/formdata)),
* **response.blob()** – return the response as [Blob](https://javascript.info/blob) (binary data with type),
* **response.arrayBuffer()** – return the response as [ArrayBuffer](https://javascript.info/arraybuffer-binary-arrays) (low-level representation of binary data),

A fetch() method can be used with many type of reuqests such as **POST**, **GET**, **PUT**and **DELETE**.

**GET method using fetch API:**

If the contents of the response are in the raw text format, you can use the text() method. The text() method returns a Promise that resolves with the complete contents of the fetched resource:

fetch('/readme.txt')

.then(response => response.text())

.then(data => console.log(data));

**Ex:2**fetch('https://www.reddit.com/r/javascript/top/.json?limit=5')

.then(res => console.log(res));

If you inspect the response in your browser’s console, you should see a Response object with several properties:

{

body: ReadableStream

bodyUsed: false

headers: Headers {}

ok: true

redirected: false

status: 200

statusText: ""

type: "cors"

url: "https://www.reddit.com/top/.json?count=5"

}

We can’t block the user interface waiting until the request finishes. That’s why fetch() returns a Promise, an object which represents a future result. In the above example, we’re using the then method to wait for the server’s response and log it to the console.

Now let’s see how we can extract the JSON payload from that response once the request completes:

fetch('https://www.reddit.com/r/javascript/top/.json?limit=5')

.then(res => res.json())

.then(json => console.log(json));

We start the request by calling fetch(). When the promise is fulfilled, it returns a Response object, which exposes a json method. Within the first then() we can call this json method to return the response body as JSON.

However, the json method also returns a promise, which means we need to chain on another then(), before the JSON response is logged to the console.

**POST request using fetch API:**  
The post request is widely used to submit forms to the server. Fetch also supports the POST method call. To do a POST request we need to specify additional parameters with the request such as **method**, **headers**, etc.  
In this example, we’ll do a POST request on the same JSONPlaceholder and add a post in the posts. It’ll then return the same post content with an ID.

In the same JavaScript file add the following content:

|  |
| --- |
| //  main.js    // POST request using fetch()  fetch("<https://jsonplaceholder.typicode.com/posts>", {        // Adding method type      method: "POST",        // Adding body or contents to send      body: JSON.stringify({          title: "foo",          body: "bar",          userId: 1      }),        // Adding headers to the request      headers: {          "Content-type": "application/json; charset=UTF-8"      }  })    // Converting to JSON  .then(response => response.json())    // Displaying results to console  .then(json => console.log(json)); |