



# FetchAPI

---

.NET

*The Fetch API provides an interface for fetching resources that has a more powerful and flexible feature set than XMLHttpRequest.*

[HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/API/FETCH\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

# FetchAPI - Overview

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/using\\_fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/using_fetch)

---

**Fetch()** is an async method of the *WindowOrWorkerGlobalScope* ‘mixin’ (not an Interface). It provides **Request** and **Response** objects involved with network requests. **Fetch()** has built-in functions and properties that can be used when needed.

**Fetch()** includes concepts such as **CORS** and the HTTP Origin headers.

**Fetch()** requires at least one argument (the URL). It returns a **Promise** that represents the response to the request.

**Fetch()** never fails, even if the HTTP Response Code is 404 or 500. It will reject if the request itself fails (a network failure).

**Fetch()** can send and receive cross-site cookies (**Sessions**)

```
fetch('http://revature.com/assoc')  
  .then(res => res.json())  
  .then(data => console.log(data));
```

# Fetch API Example

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/using\\_fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/using_fetch)

---

```
fetch('http://revature.com/associates')  
  .then(response => response.json())  
  .then(data => console.log(data));
```

In this example, a URL is *Fetch*'ed and the response is printed to the console. `fetch()` takes one argument (there is an overload) and returns a ***Promise*** containing the ***Response*** object.

When `fetch()` returns, the ***Promise*** response becomes the parameter for the following `.then()` statement. The `.json()` method is used to extract the JSON ***Body*** content from the response.

The `.then()` statement is also used to handle whatever HTTP response codes are returned, even if 404 or 500.

# Fetch() – Arguments

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch>

---

**Fetch()** has two possible arguments.

1. The URL path to the desired resource.
2. An (optional) object called an *init* object. This allows custom settings to be set with the **Request**. (see next slide)

Below are the most frequently used *init* object properties.

Option	Usage
Method	The HTTP verb of the request. GET, POST, etc.
Headers	A Headers object containing the headers desired.
Body	An object containing what you want to POST, Delete, INSERT, etc.
Mode	The mode desired. 'cors', 'no-cors', 'same-origin'

# Fetch() – example

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/using\\_fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/using_fetch)

Below are the options available for each property.

**\* == default**

The path to the resource –

The init object –

**\*GET**, POST, PUT, DELETE, etc –

'no-cors', **\*'cors'**, 'same-origin' –

**\*default**, no-cache, reload, force-cache, only-if-cached –

include, **\*same-origin**, omit –

'application/json', 'application/x-www-form-urlencoded' –

manual, **\*follow**, error –

'no-referrer', **\*'no-referrer-when-downgrade'**, 'origin',

'origin-when-cross-origin', 'same-origin', 'strict-origin',

'strict-origin-when-cross-origin', 'unsafe-url' –

Use JSON.stringify() to serialize the body. –

Body data type must match "Content-Type" header –

Use `response.json()` to parse JSON data. –

```
fetch('https://revature.com/associates',
{
  method: 'POST',
  mode: 'cors',
  cache: 'no-cache',
  credentials: 'same-origin',
  headers:
    {'Content-Type': 'application/json'},
  redirect: 'follow',
  referrerPolicy: 'no-referrer',
  body: JSON.stringify(
    { name: 'Mark', id: 42 })
})
.then(response => response.json())
.then(data => console.log(data));
```



# Promises

<https://javascript.info/promise-basics>

<https://javascript.info/promise-basics#consumers-then-catch-finally>

<https://web.dev/promises/>

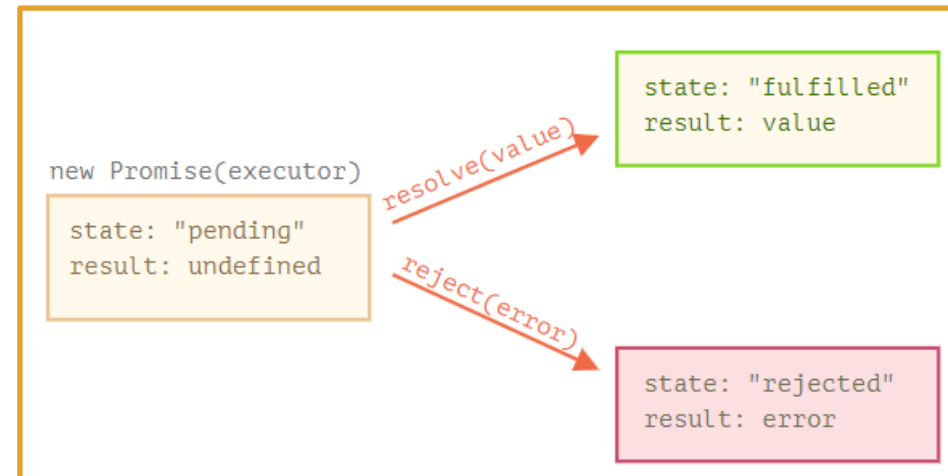
---

**Fetch()** returns a **Promise**. A **Promise** is the async result of the Fetch HTTP Request. It contains the **Fetch()** Response.

The **Fetch()** Response has either been 'resolved' or 'rejected'.

**Resolve** and **Reject** are internal JS functions so you don't have to implement them.

The '**resolved**' or '**rejected**' status of a response can be checked and acted upon with the **.then()**, **.catch()**. and/or **.finally()** blocks.



```
1 let promise = new Promise(function(resolve, reject) {
2   setTimeout(() => resolve("done!"), 1000);
3 });
4
5 // resolve runs the first function in .then
6 promise.then(
7   result => alert(result), // shows "done!" after 1 second
8   error => alert(error) // doesn't run
9 );
```

# Promises - .then(), .catch(), .finally() (1 / 3)

<https://javascript.info/promise-basics>

<https://javascript.info/promise-basics#consumers-then-catch-finally>

---

**.then()** has two arguments (**.then(a, b)**).

1. **a** - A function for a Resolved Request that contains a value.
2. **b** - A function for a Rejected/failed Request that contains an error.

```
1 let promise = new Promise(function(resolve, reject) {
2   setTimeout(() => resolve("done!"), 1000);
3 });
4
5 // resolve runs the first function in .then
6 promise.then(
7   result => alert(result), // shows "done!" after 1 second
8   error => alert(error) // doesn't run
9 );
```



# Promises - .then(), .catch(), .finally() (1 / 3)

<https://javascript.info/promise-basics>

<https://javascript.info/promise-basics#consumers-then-catch-finally>

---

**.catch()** is the equivalent of **.then(null, b)**. It will catch any errors returned in the response. It only runs if there is an error.

```
1 let promise = new Promise((resolve, reject) => {
2   setTimeout(() => reject(new Error("Whoops!")), 1000);
3 });
4
5 // .catch(f) is the same as promise.then(null, f)
6 promise.catch(alert); // shows "Error: Whoops!" after 1 second
```

# Promises - .then(), .catch(), .finally() (1 / 3)

<https://javascript.info/promise-basics>

<https://javascript.info/promise-basics#consumers-then-catch-finally>

---

**.finally()** is equivalent to **.then(c, c)** because it always runs whether the result is successful or an error. It can be placed before or after **.then()**.

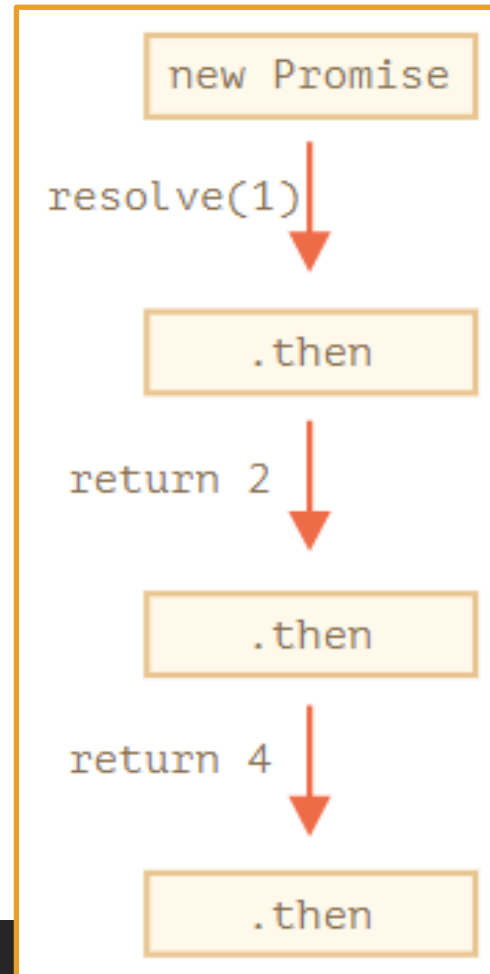
```
1 new Promise((resolve, reject) => {  
2   /* do something that takes time, and then call resolve/reject */  
3 })  
4 // runs when the promise is settled, doesn't matter successfully or not  
5 .finally(() => stop loading indicator)  
6 .then(result => show result, err => show error)
```

# Promise Chaining

<https://javascript.info/promise-chaining>

**Promise Chaining** is a way to run multiple actions on the returns of sequential asynchronous functions.

The result of each function is passed through the chain of **.then()** handlers.



```
fetch('https://revature.com/associates')
  .then(response1 => response.json())
  .finally(response2 => console.log('This is
    the finally block. It always runs.'))
  .then(response3 => console.log('This is
    the second .then() block'))
  .then(response4 => {
    console.log('This is the third .then()
      block');
    resolve();
  })
  .catch(response5 => console.log('The
    catch block won't run unless
    there's an error'));
```

# Fetch() – Response Object

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/using\\_fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/using_fetch)

---

A **Response** object is returned when a **fetch()** *Promise* is resolved. The three most commonly used response properties are:

- **Response.status** — An integer (default value 200) containing the response status code.
- **Response.statusText** — A string (default value "OK"), which corresponds to the HTTP status code message.
- **Response.ok** — This is a shorthand for checking that status is in the range 200-299 inclusive. This returns a Boolean.

```
1 fetch('flowers.jpg')
2   .then(response => {
3     if (!response.ok) {
4       throw new Error('Network response was not ok');
5     }
6     return response.blob();
7   })
8   .then(myBlob => {
9     myImage.src = URL.createObjectURL(myBlob);
10  })
11  .catch(error => {
12    console.error('There has been a problem with your fetch operation:', error);
13  });
```

# Fetch() – checking Response success

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/using\\_fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/using_fetch)

A **fetch()** promise will reject with a **TypeError** when a network error is encountered or **CORS** is misconfigured on the server-side.

A check for a successful **fetch()** includes checking that the **Promise** 'resolved'.  
Checking that the **Response.ok** property has a value of true.

1. With **response.ok**, check that the **Response** was completed and take action based on the data received.
2. Use **.catch()** to handle any errors that could have been thrown.

```
1 fetch('flowers.jpg')
2   .then(response => {
3     if (!response.ok) {
4       throw new Error('Network response was not ok');
5     }
6     return response.blob();
7   })
8   .then(myBlob => {
9     myImage.src = URL.createObjectURL(myBlob);
10  })
11  .catch(error => {
12    console.error('There has been a problem with your fetch operation:', error);
13  });
```

# Fetch() – Body

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)

---

Both requests and responses may contain **body** data. A **body** is an instance of any of the following types.

Data Type	Function used to access data
ArrayBuffer	<b>.arrayBuffer()</b> - This object is used to represent a generic, fixed-length binary data buffer.
ArrayBufferView	
Blob/File	<b>.blob()</b> - returns a promise that resolves with a Blob.
String	<b>.text()</b> - returns a promise that resolves with a USVString object (text). The response is always decoded using UTF-8.
FormData	<b>.formData()</b> - returns a promise that resolves with a FormData object.