Fetch API - Request

In Fetch API, Request interface is used to create a resource request. It is an alternative way of creating requests other than the fetch() function. It also provides various properties and methods which we can apply to the request. So, first, we will learn about Request() constructor, then how to send requests and then the method and properties provided by the Request interface.

Constructor

To create a request object we can use Request() constructor along with a new keyword. This constructor contains one mandatory parameter which is the URL of the resource and the other parameter is optional.

Syntax

```
const newRequest = new Request(resourceURL)
Or
const newRequest = new Request(resourceURL, optional)
```

The Request() constructor has the following parameters −

- **resourceURL** − The resource which we want to fetch. Its value can be either a resource URL or the Request object.
- **Options** − Object which provides additional settings for the request and the customized options are as follows −
  - **method** − Represents the request methods like GET, POST, PUT and DELETE.
  - **headers** − Set a header to the request.
  - **body** − Adding data to the request. This parameter is not used by GET or HEAD methods.
  - **mode** − Set the mode for the request such as cors, same-origin, no-cors or navigate. By default the value of the mode parameter is cors.
  - **credentials** − It sets the credentials which you want to use for the request such as omit, same-origin, or include. The default value of this parameter is same-origin.
  - **cache** − Set the cache mode you want for your request.

- **redirect** – Used for redirect mode such as follow, error, or manual. By default, the parameter is set for follow value.
- **referrer** – A string which represents the referrer of the request such as client, URL, or no-referrer. The default value of this parameter is about the client.
- **referrerPolicy** – Used to set the referrer policy.
- **integrity** – Used to set the subresource integrity value of the given request.
- **keepalive** – Used to check whether to create a persistent connection for multiple requests/response or not.
- **signal** – Represent an AbortSignal object which is used to communicate with or abort a request.
- **priority** – Used to set the priority of the request as compared to other requests. The possible value of this parameter is:

- **high** – Set the priority of the current fetch request to high as compared to others.
- **low** – Set the priority of the current fetch request to low as compared to others.
- **auto** – Automatically find the priority of the current fetch request.

Explore our latest online courses and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

Send Request

To send a request, we must first create a Request object using the Request constructor with additional parameters like header, body, method, resource URL, etc. Then pass this object in the fetch() function to send the request to the server. Now the fetch() function returns a promise which will resolve with the response object. If we encounter an error, then we execute the catch block.

Example

In the following program, we create a script to send data using the Request object. So for that, we create a request object using Request() constructor along with parameters like –

- **URL** – Represent the resource URL.
- **method** – Here we use the POST method which represents we are sending data to the server.

- **body** – Contains the data which we want to send.
- **header** – It tells that the data is JSON data.

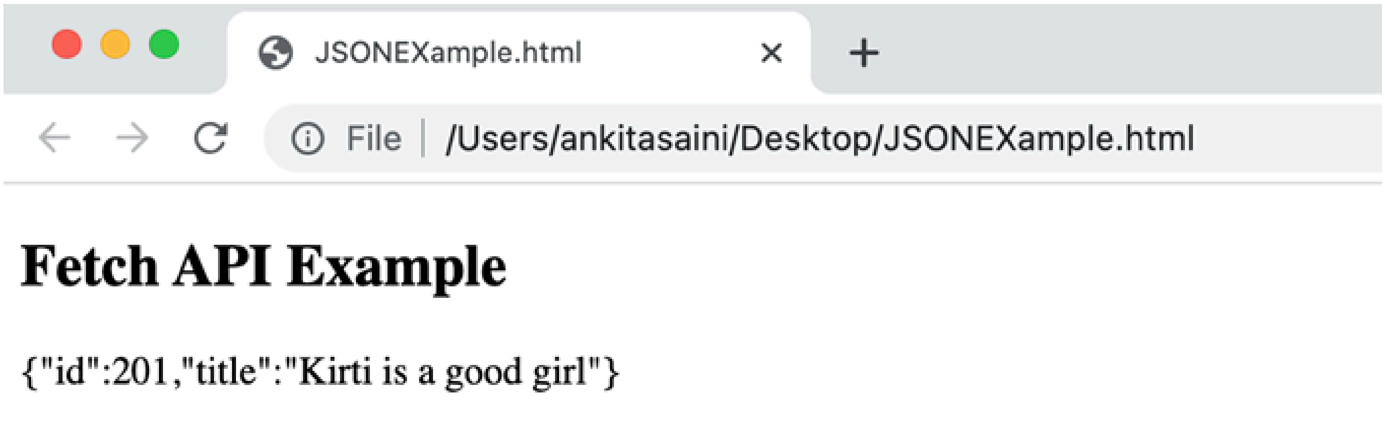
Now we pass the request object in the fetch() function to send the request and handle the response returned by the server and handle the error if it occurs.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Creating request object
   const myRequest = new Request("https://jsonplaceholder.typicode.com/todos", {
      // Setting POST request
      method: "POST",

      // Add body which contains data
      body: JSON.stringify({
      id: 321,
      title: "Kirti is a good girl",
      }),

      // Setting header
      headers:{"Content-type": "application/json; charset=UTF-8"}
   });
   fetch(myRequest)

   // Handling response
   .then(response => response.json())
   .then(myData => {
      console.log("Data Sent Successfully");
      // Display output
      document.getElementById("sendData").innerHTML = JSON.stringify(myData);
   })

   // Handling error
   .catch(err=>{
      console.error("We get an error:", err);
   });
</script>
   <h2>Fetch API Example</h2>
```

```
      <div>
         <!-- Displaying retrieved data-->
         <p id="sendData"></p>
      </div>
   </body>
```

Output



Fetch API Example

```
{"id":201,"title":"Kirti is a good girl"}
```

Instance Properties

The properties provided by the request interface are the read-only properties. So the commonly used properties are –

Sr.No.	Property & Description
1	**Request.url** This property contains the URL of the given request.
2	**Request.body** This property contains the body of the given request.
3	**Request.bodyUsed** This property is used to tell whether the body present in the request is used or not. Its value is boolean.
4	**Request.destination** This property is used to tell the destination of the request.

5	**Request.method** This property contains the request methods such as GET, POST, PUT, and DELETE.
6	**Request.headers** This property contains the header object of the request.
7	**Request.cache** This property contains the cache mode of the given request.
8	**Request.credentials** This property contains the credentials of the given request.
9	**Request.mode** This property contains the mode of the given request.

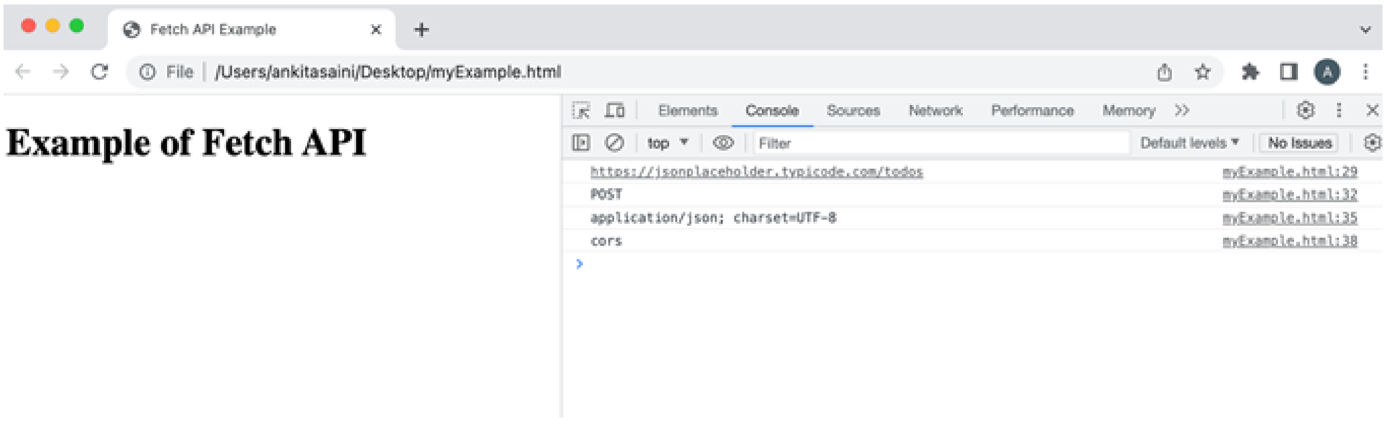
Example

In the following program, we use the properties (such as url, method, headers, and mode)provided by the Request interface.

Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <title>Fetch API Example</title>
</head>
<body>
   <h1>Example of Fetch API</h1>
<script>
   // Creating request object
   const myRequest = new Request("https://jsonplaceholder.typicode.com/todos", {
      // Setting POST request
      method: "POST",

      // Add body which contains data
      body: JSON.stringify({
         id: 321,
         title: "Kirti is a good girl",
      }),
      // Setting header
      headers:{"Content-type": "application/json; charset=UTF-8"},
      mode: "cors"
   });
   // Display url of the request
   console.log(myRequest.url);

   // Display request method
   console.log(myRequest.method);

   // Display header of the request
   console.log(myRequest.headers.get('content-Type'));

   // Display mode of the request
   console.log(myRequest.mode);
</script>
</body>
</html>
```

Output



Example of Fetch API

```
https://jsonplaceholder.typicode.com/todos          myExample.html:29
POST                                                myExample.html:32
application/json; charset=UTF-8                     myExample.html:35
cors                                                myExample.html:38
```

Methods

The following are the commonly used method of Request interface −

Sr.No.	Method & Description
1	**Request.arrayBuffer()** This method is used to resolve a promise with ArrayBuffer representation of the request body.
2	**Request.blob()** This method is used to resolve a promise with a blob representation of the request body.

3	**Request.clone()** This method is used to create a copy of the current request.
4	**Request.json()** This method is used to parse the request body as JSON and resolve a promise with the result of parsing.
5	**Request.text()** This method is used to resolve a promise with a text representation of the request body.
6	**Request.formData()** This method is used to resolve a promise with formData representation of the request body.

Example

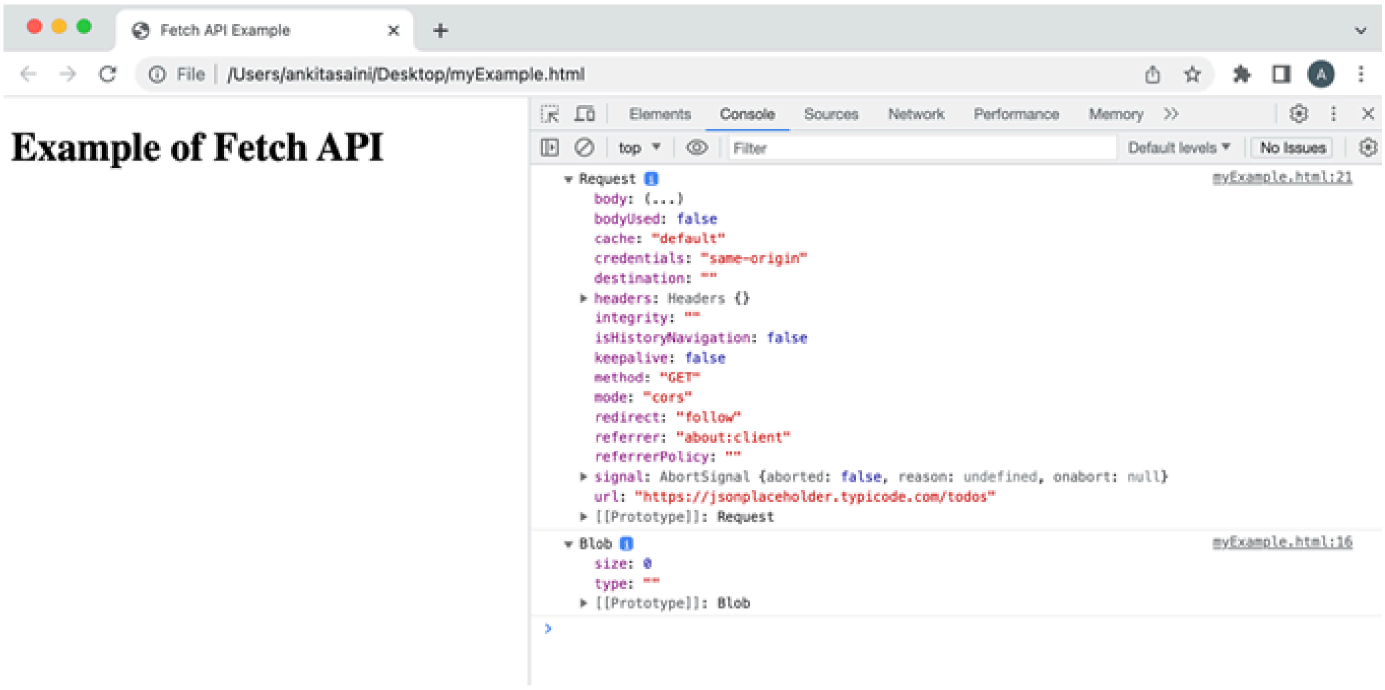
In the following program, we use the methods(such as blob, clone, etc) provided by the Request interface.

Open Compiler

```
<!DOCTYPE html>
<html>
<head>
   <title>Fetch API Example</title>
</head>
<body>
   <h1>Example of Fetch API</h1>
<script>
   // Creating request object
   const myRequest = new Request("https://jsonplaceholder.typicode.com/todos");

   // Using blob() method
   myRequest.blob()
   .then(data =>{
      console.log(data)
   });

   // Creating a copy of the request using the clone() method
   const duplicate = myRequest.clone();
   console.log(duplicate);
</script>
```

```
</body>
</html>
```

Output



Example of Fetch API

```
▾ Request                                                       myExample.html:21
    body: (...)
    bodyUsed: false
    cache: "default"
    credentials: "same-origin"
    destination: ""
  ▸ headers: Headers {}
    integrity: ""
    isHistoryNavigation: false
    keepalive: false
    method: "GET"
    mode: "cors"
    redirect: "follow"
    referrer: "about:client"
    referrerPolicy: ""
  ▸ signal: AbortSignal {aborted: false, reason: undefined, onabort: null}
    url: "https://jsonplaceholder.typicode.com/todos"
  ▸ [[Prototype]]: Request
▾ Blob                                                          myExample.html:16
    size: 0
    type: ""
  ▸ [[Prototype]]: Blob
>
```

Conclusion

So this is how the Request interface works in Fetch API. It provides various ways to construct and customize the request. Or we can say that it provides flexibility and more control over the request. Now in the next article, we will see how the Response interface is used in the Fetch API.