Fetch API - Response

Fetch API provides a special interface to create a response to a request and the name of that interface is Response. This interface provides a Response() constructor to create a response object. It provides various methods and properties to access and handle response data.

Constructor

To create a response object we can use the Response() constructor along with a new keyword. This constructor may or may not contain parameters.

Syntax

```
const newResponse = New Response()
Or
const newResponse = New Response(rBody)
Or
const newResponse = New Response(rBody, rOption)
```

The Response() constructor has the following parameters –

- **rBody** – It represents an object which defines a body for the response. Its value can be null(default value) or blob, ArrayBuffer, TypedArray, DataView, FormData, String, URLSearchParams, a string literal, or ReadableStream.
- **Options** – It is an object which is used to provide customised settings which we want to apply to the response and the options are:
- **headers** – It is used to add a header to your response. By default the value of this parameter is empty. Its value can be a Header object or an object literal of string.
- **status** – This parameter represents the status code for the response. Its default value is 200.
- **statusText** – This parameter represent a status message related to the status code like "Not Found". Its default value is "".

Example

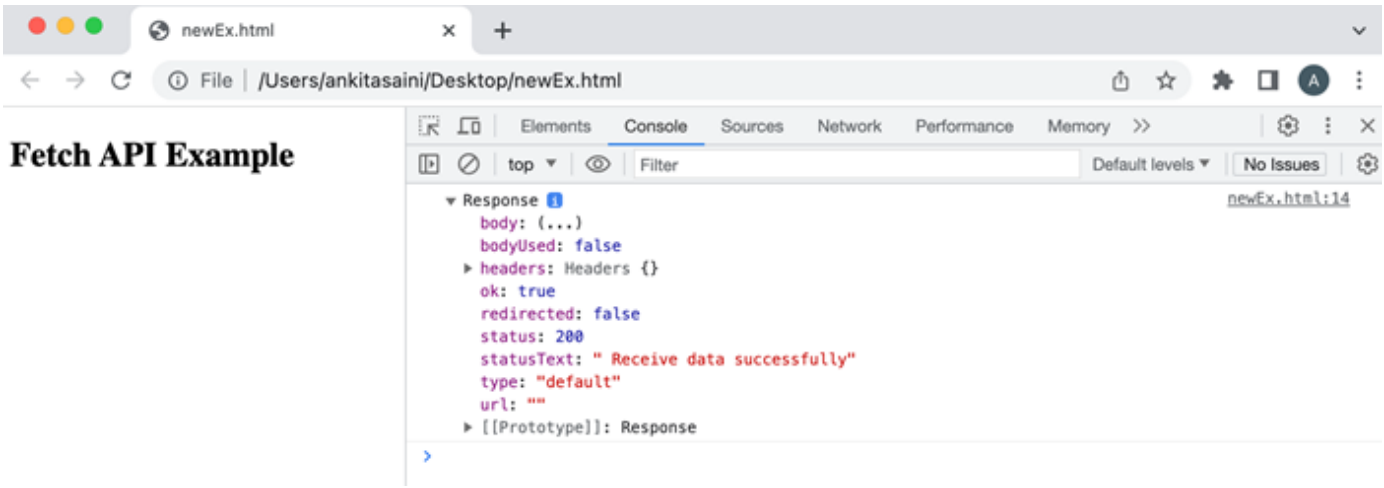
In the following program, we fetch the data from the given URL using the fetch() function and then display the response data in JSON format.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Data
   const option = {message: "Hello Tom. How are you?"};

   // creating header object
   const newResponse = new Response(JSON.stringify(option), {
      status: 200,
      statusText:" Receive data successfully"
   });
   // Displaying response
   console.log(newResponse)
</script>
   <h2>Fetch API Example</h2>
   <div>
      <!-- Displaying retrieved data-->
      <p id="sendData"></p>
   </div>
</body>
</html>
```

Output



Instance Properties

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

Sr.No.	Property & Description
1	**Response.body** This property contains a ReadableStream body content.
2	**Response.ok** This property checks whether the response was successful or not. The value of this property is in boolean.
3	**Response.bodyUsed** This property is used to check whether the body is used in the response or not. Its value is boolean.
4	**Response.redirected** This property is used to check whether the response is the result of the redirect or not. Its value is in boolean.
5	**Response.status** This property contains the status code of the response.
6	**Response.statusText** This property provides the status message according to the status code.
7	**Response.type** This property provides the type of response.
8	**Response.url** This peoperty provides the url of the response.
9	**Response.header**

	This property provides the Header object of the given response.

Example

In the following program, we use the properties provided by the Response interface.

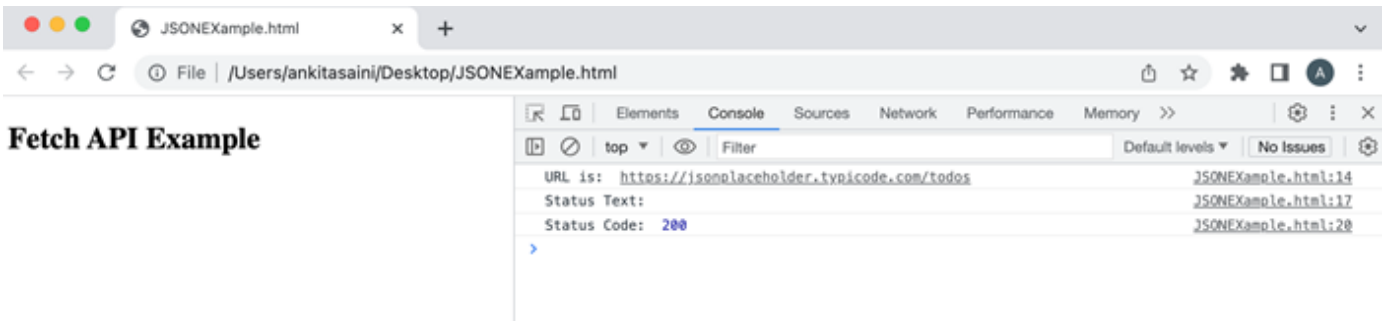
</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
   <h2>Fetch API Example</h2>
<script>
   // GET request using fetch()function
   fetch("https://jsonplaceholder.typicode.com/todos")
   .then(response => {
   // Finding response URL
   console.log("URL is: ", response.url);

   // Getting response text
   console.log("Status Text: ", response.statusText);

   // Getting response status
   console.log("Status Code: ", response.status);
   }).catch(err =>{
      // Handling error
      console.log("Found Error:", err);
   });
</script>
</body>
</html>
```

Output



Methods

The following are the commonly used method of Response interface –

Sr.No.	Method & Description
1	**Request.arrayBuffer()** This method is used to return a promise which will resolve with the ArrayBuffer representation of the response body.
2	**Request.blob()** This method is used to return a promise which will resolve with Blob representation of the response body.
3	**Request.clone()** This method is used to create a copy of the current response object.
4	**Request.json()** This method is used to parse the response body as JSON and return a promise that will resolve with the result of parsing.
5	**Request.text()** This method is used to return a promise which will resolve with a text representation of the response body.
6	**Request.formData()** This method is used to return a promise which will resolve with a FormData representation of the response body.
7	**Response.error()** This method returns a new Response object related to a network error. It is a static method.
8	**Response.redirect** This method returns a new Response object with a different URL. It is a static method.
9	**Response.json()** This method returns a new Response object for the returning JSON encoded data. It is a static method.

Example

In the following program, we use the methods provided by the Response interface. So here we are going to use the json() function to parse the response in the JSON format.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // GET request using fetch()function
   fetch("https://jsonplaceholder.typicode.com/todos/2", {
   // Method Type
   method: "GET"})

   // Parsing the response data into JSON
   // Using json() function
   .then(response => response.json())
   .then(myData => {
      // Display output
      document.getElementById("manager").innerHTML = JSON.stringify(myData);
   }).catch(newError =>{
      // Handling error
      console.log("Found Error:", newError)
   });
</script>
   <h2>Display Data</h2>
   <div>
      <!-- Displaying retrevie data-->
      <table id = "manager"></table>
   </div>
</body>
</html>
```

Output

Display Data

{"userId":1,"id":2,"title":"quis ut nam facilis et officia qui","completed":false}

Conclusion

So this is how the Response interface works with fetch API. Using the Response interface we can extract and process the response provided by the server. It also provides various methods and properties for extracting and processing the response. Now in the next article, we will learn about body data in the fetch API.