Fetch API - Send Data Objects

In Fetch API, we can send data objects from a web browser to a web server. A data object is an object which contains data in the key-value or property-value pair. Or we can say that a data object is data which we add in the request body while creating an HTTP request using Fetch API.

Fetch API supports various data formats; you can choose them according to the content type header you set or the server's requirement. Some of the commonly used data formats are −

JSON

JSON is known as JavaScript Object Notation. It is the most commonly used data format to exchange data between the web browser and the server. In JSON format, the data is stored in the form of key-value pairs and provide full support to nested objects or arrays. To send data in the JSON format we need to convert JavaScript objects into JSON strings with the help of the "JSON.stringfy()" function.

Following is the JSON format of data −

```
const newData = {
    empName: "Pooja",
    empID: 2344,
    departmentName: "HR"
};
```

Where "empName", "empID", and "department" are the keys and "Pooja", "2344", and "HR" are their values.

The following headers are used for JSON format −

```
headers:{"Content-type": "application/json; charset=UTF-8"}
```

It tells the server that the received data is in JSON format.

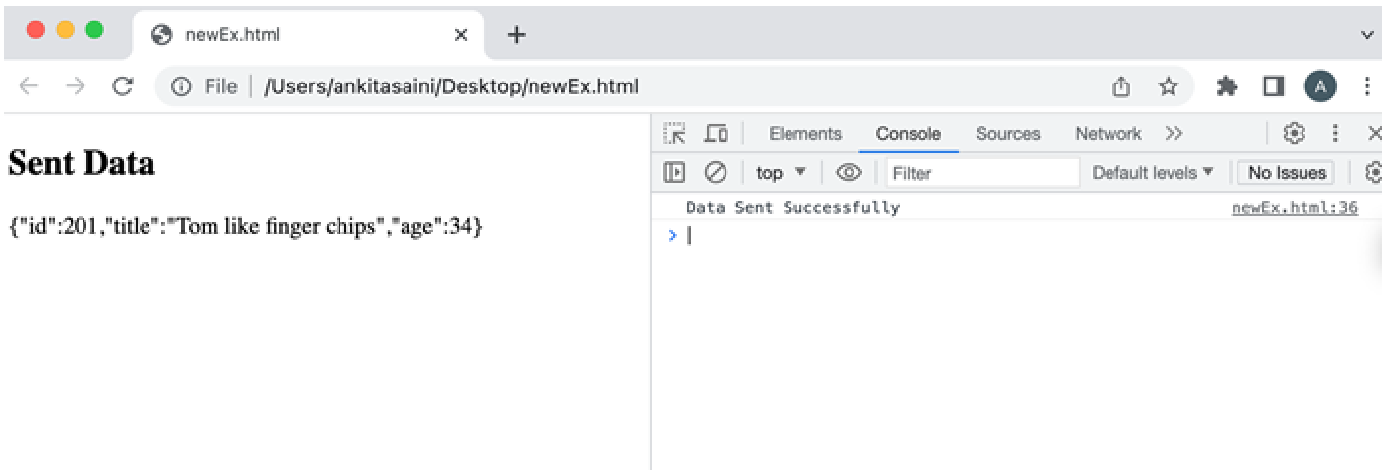
Example

In the following program, we create a script to send data in JSON format. So for that, we create a data object with key-value pairs. Now we use the fetch() function to send request to the server. In this fetch function, we include the request method that is "POST", set the header to "application/json" which tells the server that the send data is in JSON, and include the data object in the body of the request by converting into JSON string using "JSON.stringify()" function. After sending the request to the server now we use the then() function to handle the response. If we encounter an error, then that error is handled by the catch() function.

Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Data object
   const newData = {
      id: 45,
      title: "Tom like finger chips",
      age: 34
   };
   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Adding POST request to send data
      method: "POST",

      // Adding header
      headers:{"Content-type": "application/json; charset=UTF-8"},

      // Adding body which we want to send
      // Here we convert data object into JSON string
      body: JSON.stringify(newData)
   })
   // Converting received information into JSON
   .then(response =>{
      if (response.ok){
         return response.json()
      }
   })
   .then(myData => {
      // Display result
      console.log("Data Sent Successfully");

      // Display output
      document.getElementById("sendData").innerHTML = JSON.stringify(myData);
   }).catch(err=>{
      console.log("Found error:", err)
   });
</script>
   <h2>Sent Data</h2>
   <div>
      <!-- Displaying data-->
      <p id = "sendData"></p>
   </div>
</body>
</html>
```

Output


newEx.html
File | /Users/ankitasaini/Desktop/newEx.html
Elements Console Sources Network
top Filter Default levels No Issues

Sent Data

{"id":201,"title":"Tom like finger chips","age":34}

```
Data Sent Successfully                    newEx.html:36
```

FormData

FormData is an in-built JavaScript object. It is used to send data in the HTML form format. In FormData, we can store data in the form of key-value pairs where the key represents the field of the form and the value represents the value of that field. It can handle binary data, files, and other form types. To create a new form object we need to use FormData() constructor along with a new keyword.

Syntax

```
const newform = new FormData()
```

The append() function is used to add new key-value pair in the FormData object.

Syntax

```
newform.append("name", "Mohina");
```

Where "name" is the key or field of the form and "Mohina" is the value of the field. While working with FormData objects in Fetch API we do not need to set a header because Fetch API will automatically set headers for the FormData object.

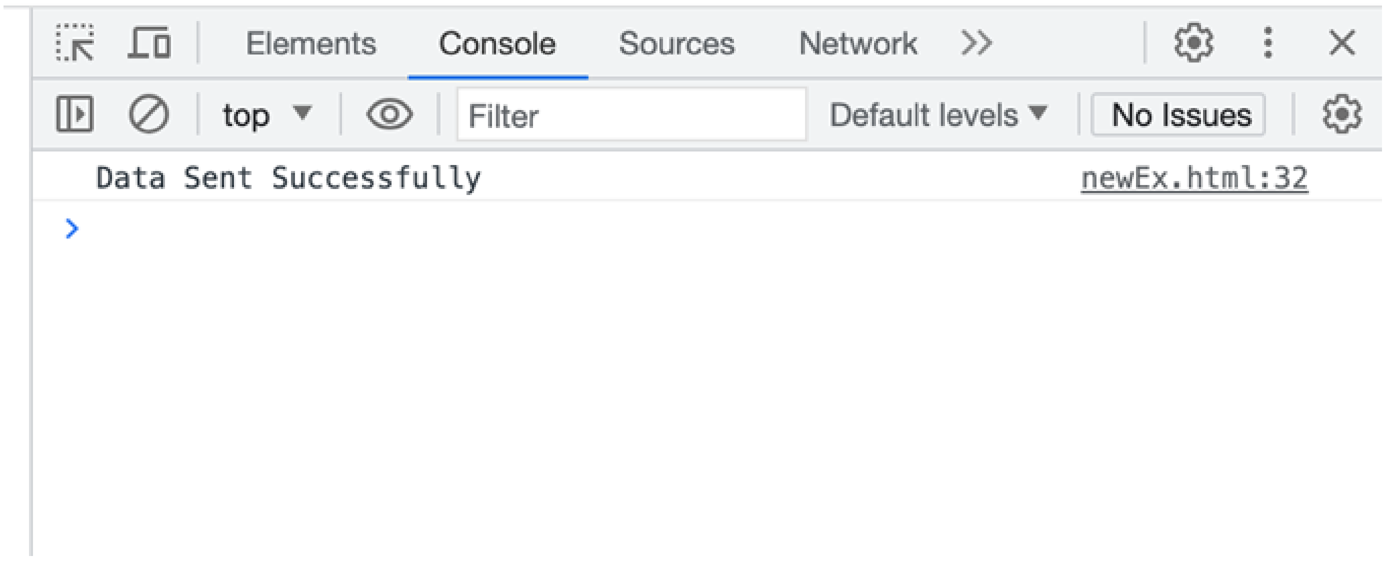
Example

In the following program, we create a script to send data in FormData. So for that, we create a FormData object using FormData() constructor and then add key-value pairs in the FormData object using the append() function. Now we use the fetch() function to send a request to the server. In this fetch function, we include the request method that is "POST" and include the FormData object in the body parameter. After sending the request to the server now we use the then() function to handle the response. If we encounter an error, then that error is handled by the catch() function.

Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // FormData object
   const newform = new FormData();

   // Adding key-value pairs in FormData object
   newform.append("id", 4532);
   newform.append("title", "Today is raining");

   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Adding POST request to send data
      method: "POST",

      // Adding body which we want to send
      // Here we add FormData object
      body: newform
   })
   // Converting received information into JSON
   .then(response =>{
      if (response.ok){
         return response.json()
      }
   })
   .then(myData => {
      // Display result
      console.log("Data Sent Successfully");

      // Display output in HTML page
      document.getElementById("sendData").innerHTML = JSON.stringify(myData);
   }).catch(err=>{
      console.log("Found error:", err)
   });
</script>
   <h2>Sent Data</h2>
   <div>
      <!-- Displaying data-->
      <p id = "sendData"></p>
   </div>
</body>
</html>
```

Output



```
Data Sent Successfully                                        newEx.html:32
>
```

Plain Text

In Fetch API, we can also send data in simple plain text. If we want to send raw text or non-standard data formats, then we send data using Plain text. To send plain text we need to simply add the text in the form of string in the body of the request.

Following is the plain text object –

```
const newData = "My car is running very fast"
```

The following headers are used for plain text –

```
headers:{"Content-type": "text/plain"}
```

It indicates to the server that the received data is in plain text.

Example

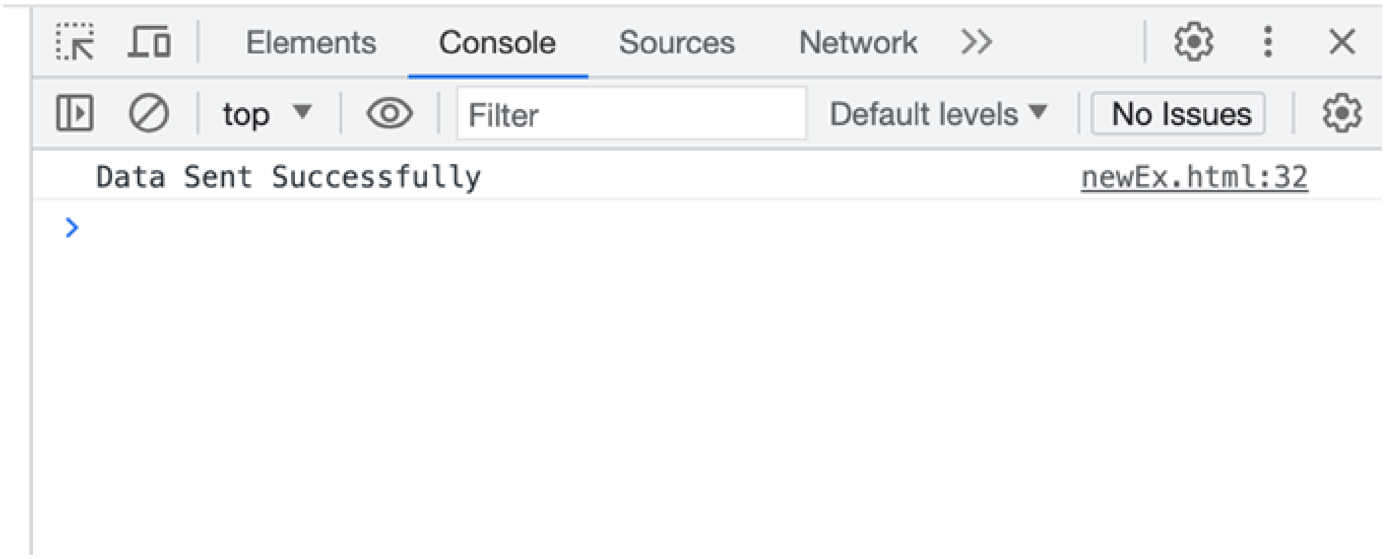
In the following program, we create a script to send data in plain text. So for that, we create a data object and assign a string value to it in simple text. Now we use the fetch() function to send a request to the server. In this fetch function, we include the request method that is "POST", set the header to "text/plain" which tells the server that the sent data is in plain text, and includes the data object in the body of the request. After sending the request to the server now we use the then() function to handle the response. If we encounter an error, then that error is handled by the catch() function.

Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // FormData object
   const newform = new FormData();

   // Adding key-value pairs in FormData object
   newform.append("id", 4532);
   newform.append("title", "Today is raining");

   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Adding POST request to send data
      method: "POST",

      // Adding body which we want to send
      // Here we add the FormData object
      body: newform
   })
   // Converting received information into JSON
```

```
    .then(response =>{
       if (response.ok){
          return response.json()
       }
    })
    .then(myData => {
       // Display result
       console.log("Data Sent Successfully");

       // Display output in HTML page
       document.getElementById("sendData").innerHTML = JSON.stringify(myData);
    }).catch(err=>{
       console.log("Found error:", err)
    });
</script>
    <h2>Sent Data</h2>
    <div>
       <!-- Displaying data-->
       <p id = "sendData"></p>
    </div>
</body>
</html>
```

Output



```
Data Sent Successfully                    newEx.html:32
```

URL-encoded Data

The URL-encoded data is the most commonly used data format to send form data in URL parameters or the body of the POST request. It represents data in the form of key-value pairs where the values are encoded with the help of percent-encoding. We can create URL-encoded data objects with the help of URLSearchParams class.

Syntax

```
const newData = new URLSearchParams()
```

The append() function is used to add new key-value pair in the URL-encoded data object.

Syntax

```
newform.append("name", "Mohina");
```

Where "name" is the key or field of the form and "Mohina" is the value of the field.

The following headers are used for URL-encoded data −

```
headers:{"Content-type": "text/plain"}
```

It indicates to the server that the received data is URL-encoded data.

Example

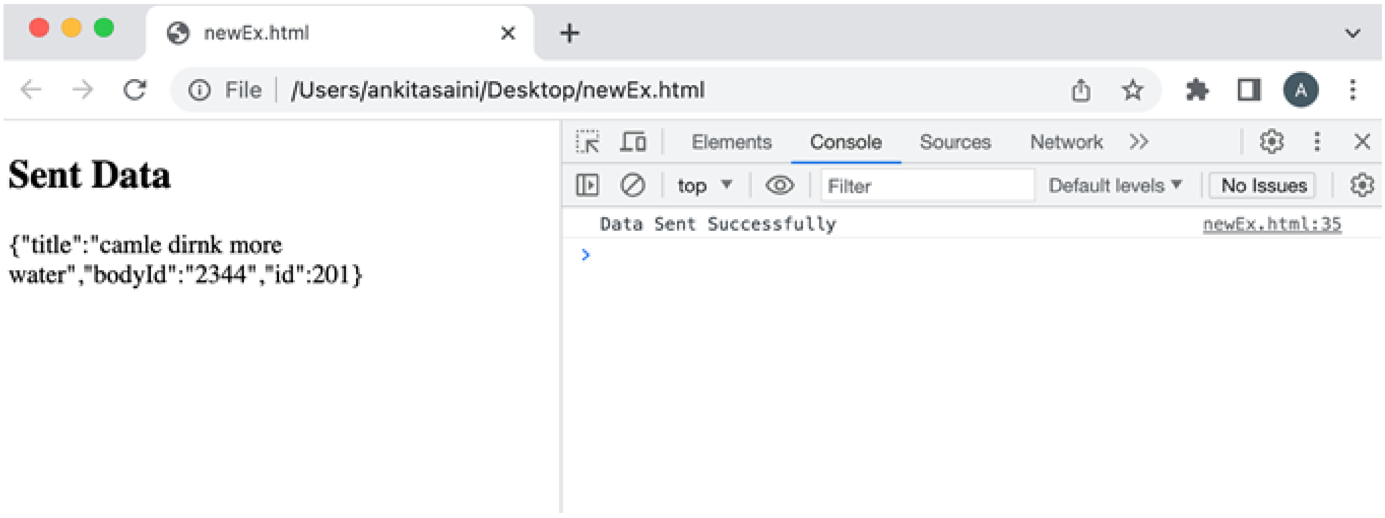
In the following program, we create a script to send data in plain URL-encoded. So for that, we create a data object using URLSearchParams() and assign key-value pairs using the append() function. Now we use the fetch() function to send a request to the server. In this fetch function, we include the request method that is "POST", set the header to "application/x-www-form-urlencoded" which tells the server that the send data is in URL-encoded format, and includes the data object in the body of the request. After sending the request to the server now we use the then() function to handle the response. If we encounter an error, then that error is handled by the catch() function.

Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // FormData object
   const newform = new FormData();

   // Adding key-value pairs in FormData object
```

```
    newform.append("id", 4532);
    newform.append("title", "Today is raining");

    fetch("https://jsonplaceholder.typicode.com/todos", {
       // Adding POST request to send data
       method: "POST",

       // Adding body which we want to send
       // Here we add FormData object
       body: newform
    })

    // Converting received information into JSON
    .then(response =>{
       if (response.ok){
          return response.json()
       }
    })
    .then(myData => {
       // Display result
       console.log("Data Sent Successfully");

       // Display output in HTML page
       document.getElementById("sendData").innerHTML = JSON.stringify(myData);
    }).catch(err=>{
       console.log("Found error:", err)
    });
    </script>
 <h2>Sent Data</h2>
 <div>
    <!-- Displaying data-->
    <p id = "sendData"></p>
 </div>
 </body>
 </html>
```

Output



Sent Data

{"title":"camle dirnk more water","bodyId":"2344","id":201}

```
Data Sent Successfully                                      newEx.html:35
>
```

Conclusion

So this is how we can send different types of data objects using Fetch API. Out of all these formats, the most commonly used formats are JSON and FormData. Also, the choice of choosing data object formats is dependent upon the requirement of the server or the type of data we want to send. So now in the next article, we will learn Cross-Origin Requests.