Fetch API - Send JSON Data

Fetch API is used to send or receive data asynchronously without refreshing the web page. In the Fetch API, we can send data in various formats like JSON, URL-encoded form, Text, FormData, Blob or ArrayBuffer. Among all these forms JSON (JavaScript Object Notation) data is the most commonly used format to send data using Fetch because it is simple, lightweight, and compatible with most of the programming languages. JSON data is generally created in the following format –

```
Const JSONData = {
   name: "Monika",
   Id: 293,
   Age: 32,
   City: "Pune"
};
```

Where name, id, Age, and City are the properties and Monika, 293, 32, and Pune are their values.

Fetch API generally sends JSON data as a payload in the request body or can be received in the response body. And the data is serialized as a string because it is easy to transmit data from one system to another.

While working with JSON data, Fetch API perform two main operations on JSON data –

Serializing – While sending JSON data in a request we need to convert the value into JSON string format using the "JSON.stringify()" function. This function takes an object or value as an input parameter and returns a string which represents JSON format. Due to serializing data, we can easily transmit data over the network.

Syntax

```
JSON.stringify()
```

Parsing – Parsing is a process in which we convert the JSON string received in the response back into the JavaScript object or value. This parsing of JSON data can be done by using the response.json() function. This function takes the response object as an argument and returns a promise which is resolved in the parsed JSON data or JavaScript object.

Syntax

```
response.json()
```


Send JSON Data

To send JSON data Fetch API uses the following ways –

- Using fetch() function
- Using the fetch() function with async/await
- Using request object

Method 1 – Using the fetch() function

We can send data using the fetch() function. In this function, we create JSON data in the body parameter and use the POST request method to send data on the specified URL.

Example

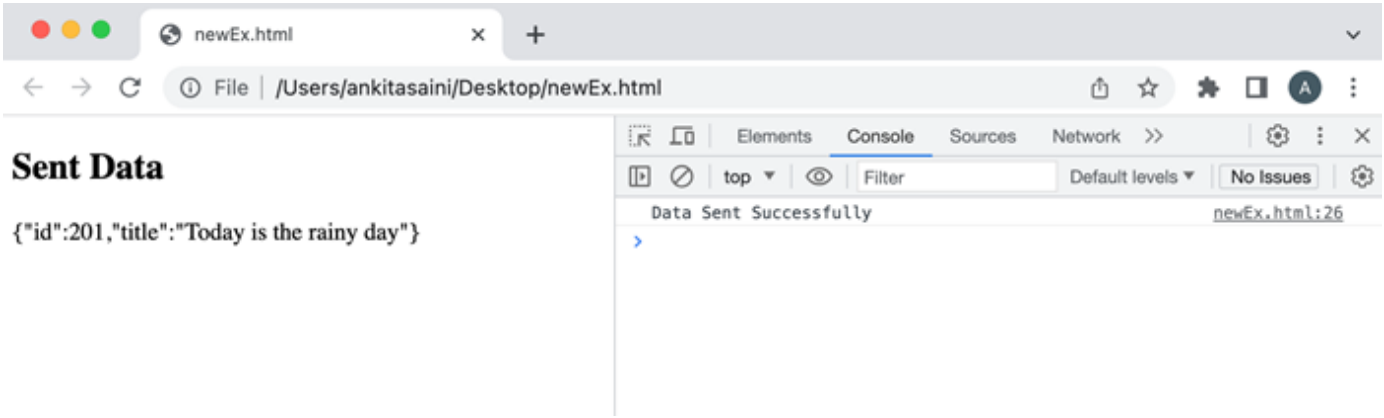
In the following program, we will send JSON data using the fetch() function. The fetch() function is used to create a request. The request contains the POST method which tells us that we want to send data, a body which contains JSON data which is converted into a string using stringify() and the header which specifies that we are sending JSON data. After sending the request the server returns a promise which will resolve to a Response object and using .json() we parse the JSON data and display the result in the console log. If we encounter an error, then the error is handled by the catch block.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Sending JSON data using a POST request
   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Setting POST request
      method: "POST",
```

```
        // Add a body which contains JSON data
        body: JSON.stringify({
            id: 290,
            title: "Today is the rainy day",
        }),
        // Setting headers
        headers:{"Content-type": "application/json; charset=UTF-8"}
    })
    // Converting response to JSON
    .then(response => response.json())
    .then(myData => {
        console.log("Data Sent Successfully");
        // Display output in HTML page
        document.getElementById("sendData").innerHTML = JSON.stringify(myData);
    })
    .catch(err=>{
        console.error("We get an error:", err);
    });
</script>
<h2>Sent Data</h2>
<div>
    <!-- Displaying data-->
    <p id = "sendData"></p>
</div>
</body>
</html>
```

Output



Method 2 – Using fetch() function with async/await

We can also send JSON data using the fetch() function with async/await. Async/await allows you to create an asynchronous program which behaves more like a synchronous program which makes it easier to learn and understand.

Example

In the following program, we will send JSON data using the fetch() function with async/await. So for that, we create an async function. In the function, we use try block which uses the fetch() function along with the resource URL, the POST request method, header, and body(JSON data in string format) parameters to send JSON data to the given URL. It also uses await keyword with the fetch() function which is used to wait for the response from the server. If the response is a success, then we parse the response return by the server using the .json() function. If the status code of the response contains an unsuccessful code the else block runs. If we encounter an error during the fetch operation, then that error is handled by the catch block.

</> Open Compiler

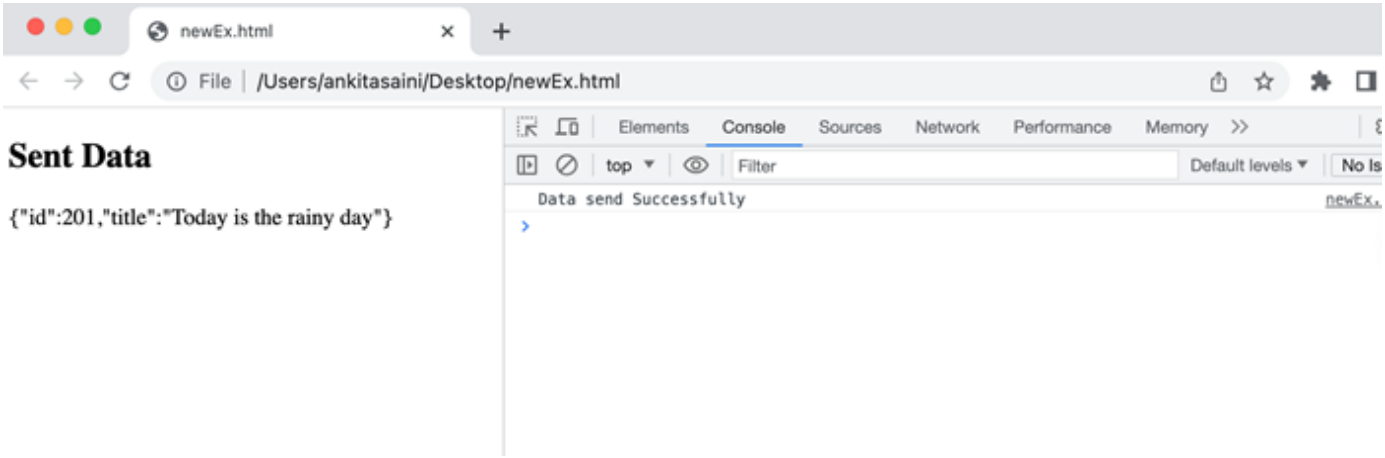
```
<!DOCTYPE html>
<html>
<body>
<script>
   async function sendingJSONData(){
      try{
         const retrunResponse = await fetch("https://jsonplaceholder.typicode.com/
            // Setting POST request to send data
            method: "POST",

            // Add body which contains JSON data
            body: JSON.stringify({
               id: 290,
               title: "Today is the rainy day",
            }),
            // Setting headers
            headers:{"Content-type": "application/json; charset=UTF-8"}
         });
         if (retrunResponse.ok){
            // Handling response
            const returnData = await retrunResponse.json();
            console.log("Data send Successfully");

            // Display output in HTML page
```

```
            document.getElementById("sendData").innerHTML = JSON.stringify(returnD
         } else {
            console.log("We found error", retrunResponse.status);
         }
      } catch(err) {
         // Handling error if occur
         console.error("Error is:", err)
      }
   }
   sendingJSONData();
</script>
<h2>Sent Data</h2>
<div>
   <!-- Displaying data-->
   <p id = "sendData"></p>
</div>
</body>
</html>
```

Output



Method 3 – Using request object

We can also send JSON data using a request object. It is an alternative to the fetch() function to send requests to the server. The request object also uses the POST method to send JSON data on the specified URL. The request object is created by using the Request() constructor of the Request interface. The request object provides more flexibility in creating or configuring the request before sending it by the fetch() function. It also allows us to add additional options like header, caching, request method, etc.

Example

In the following program, we will send JSON data using a request object. So using Request() constructor we create a request object along with parameters like resource URL, POST request method, body(JSON data in string format using stringify()), and header. Now we pass the newRequest object in the fetch function to send a request and handle the response using .then() function and parse the response using.json(). If we encounter an error during the fetch operation, then that error is handled by the catch block.

</> Open Compiler

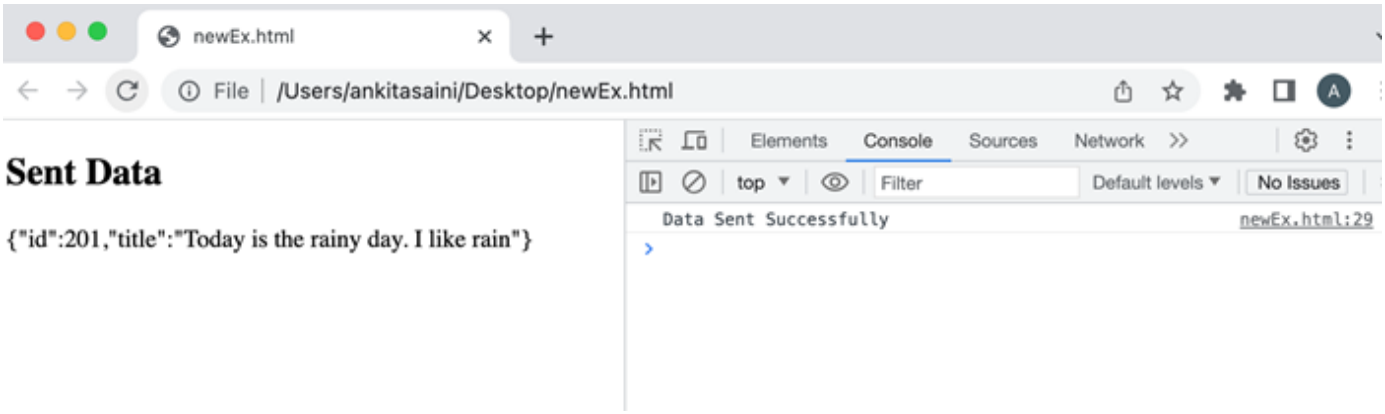
```
<!DOCTYPE html>
<html>
<body>
<script>
   // Creating request object
   const newRequest = new Request("https://jsonplaceholder.typicode.com/todos", {
      // Setting POST request
      method: "POST",

      // Add body which contains JSON data
      body: JSON.stringify({
         id: 290,
         title: "Today is the rainy day. I like rain",
      }),
      // Setting headers
      headers:{"Content-type": "application/json; charset=UTF-8"}
   });
   fetch(newRequest)
   // Handling response
   .then(response => response.json())
   .then(myData => {
      console.log("Data Sent Successfully");

      // Display output in HTML page
      document.getElementById("sendData").innerHTML = JSON.stringify(myData);
   })
   // Handling error
   .catch(err=>{
      console.error("We get an error:", err);
   });
</script>
<h2>Sent Data</h2>
<div>
```

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

Output



Conclusion

So this is how we can send JSON data in Fetch API. It is a very popular data structure in web APIs to send or receive data. It is lightweight and much more flexible as compared to other data formats. Now in the next article, we will learn how to send data objects.