Fetch API - Send PUT Requests

In the Fetch API, a PUT request is used to update or replace the existing resource or data present on the server. Using the PUT request generally contains the data which you want to update in the body of the request. When the request is received by the server, the server uses that data to update the existing resource present in the given URL. If the server does not contain the resource then it creates a new resource using the given data.

Syntax

```
fetch(URL, {
   method: "PUT",
   body: {//JSON Data},
   headers:{"content-type": "application/json; charset=UTF-8"}})
.then(info =>{
   // Code
})
.catch(error =>{
   // catch error
});
```

Here the fetch() function contains the following parameters –

- **URL** – It represents the resource which we want to fetch.
- **method** – It is an optional parameter. It is used to represent the request like, GET, POST, DELETE, and PUT.
- **body** – It is also an optional parameter. You can use this parameter when you want to add a body to your request.
- **headers** – It is also an optional parameter. It is used to specify the header.

Example 1: Sending PUT Request Using fetch()

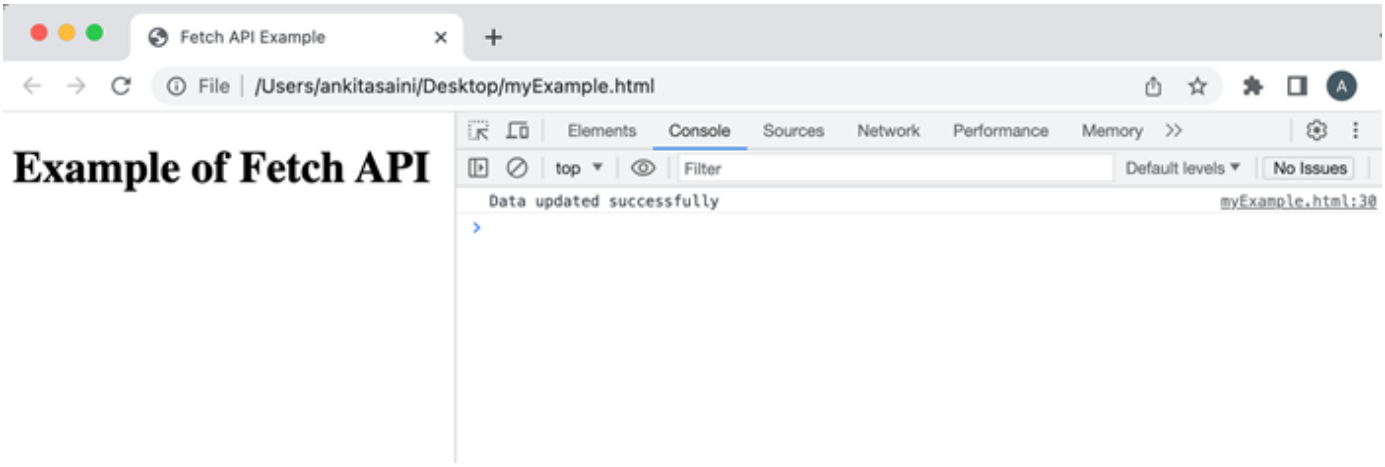
In the following program, we create a simple script to update existing data in the given URL using the PUT request using the fetch() function. Here we send a JSON document in the given URL along with the header. So after receiving the response, check the status of the response. If the response status is 200, then that means the data is updated successfully. If an error occurred, then the catch function handles that error.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
<title>Fetch API Example</title>
</head>
<body>
<h1>Example of Fetch API</h1>
<script>
   // Update data in the URL using the PUT request
   fetch("https://jsonplaceholder.typicode.com/todos/21", {
      // Using PUT request
      method: "PUT",

      // Body contains replacement data
      body: JSON.stringify({
         id: 22,
         title: "Hello! Mohina what are you doing?",
      }),
      // Setting headers
      headers:{"Content-type": "application/json; charset=UTF-8"}
   })
   .then(response => {
      // Handle response
      if (response.status == 200){
         console.log("Data updated successfully")
      } else {
         throw new error("Error Found:", response.status)
      }
   })
   // Handle error
   .catch(err=>{
      console.error(err)
   });
</script>
</body>
</html>
```

Output



Example 2: Sending PUT Request Using fetch() with async/await

In the following program, we create a script to update existing data in the given URL using the PUT request with the fetch() function and async/await. Here we send a JSON document in the given URL along with the header. So we create an async function named modifyData(). Here we use await keyword with the fetch() function to pause the execution of the function until the returned promise is resolved. After receiving the response, check the status of the response if the response status is 200, then that means the data is updated successfully. If an error occurred, then the catch function handles that error.

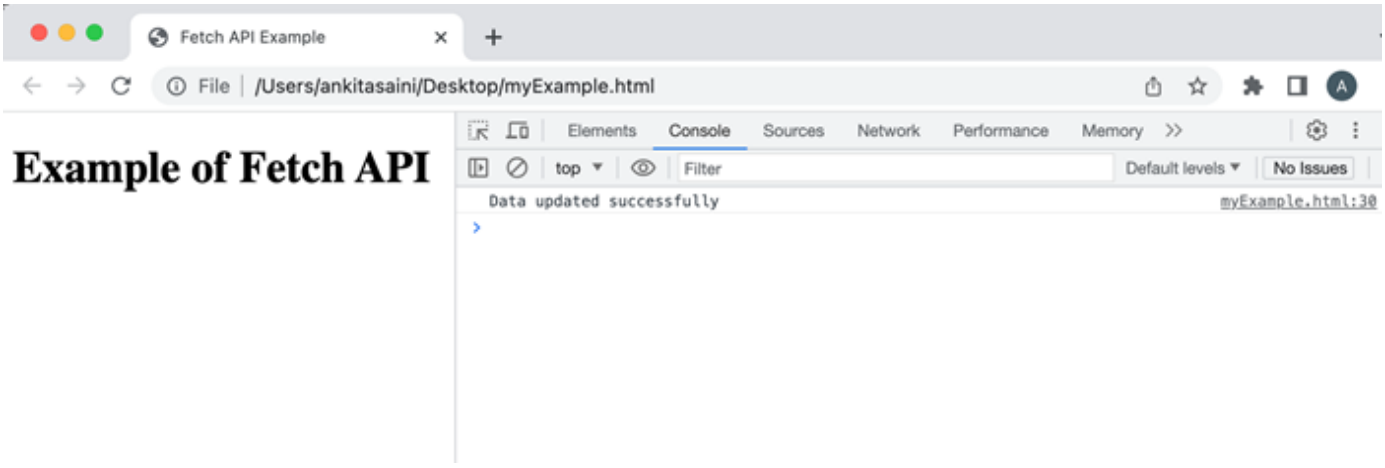
Note – Here async/ await is used together to handle asynchronous operations in a synchronous way.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<head>
<title>Fetch API Example</title>
</head>
<body>
<h1>Example of Fetch API</h1>
<script>
   async function modifyData(){
      try{
         const myRes = await fetch("https://jsonplaceholder.typicode.com/todos/21"
            // Using PUT request
            method: "PUT",
```

```
            // Body contains replacement data
            body: JSON.stringify({
               id: 24,
               title: "Mina leaves in Delhi",
            })
         });
         // Handling response
         if (myRes.status == 200){
            console.log("Data updated successfully")
         } else {
            throw new error("Error Found:", myRess.status)
         }
      } catch(err){
         console.error(err)
      }
   }
   // Calling the function
   modifyData();
</script>
</body>
</html>
```

Output



Conclusion

So this is how we can use PUT requests to update the existing data in the given resource. Using this you can also add extra properties to the request with the help of the parameters provided by the fetch() function. Now in the next chapter, we will see how we will send JSON data.