AJAX - Handling Responses

AJAX is a technique which is used to send and receive data to and from the web server asynchronously without reloading or refreshing the whole page. When an AJAX application made an asynchronous request to the server from a web page, then the server responds to the request and returns the requested data, so the receiving and handling of the server's response are known as handling responses. Or we can say that handling responses is a process that deals with the returned data from the server, performs appropriate operations on it, and updates the web page accordingly.

Handling responses covers the following points –

Receiving Response – Once the AJAX send the request to the server, then the client-side JS code waits for the server response. When the server responds to the request, the response is returned to the client.

Process Response – After getting the response from the server, now the client-side JS process the data in the expected format because the data returned by the server is in various formats like JSON, XML, etc., and also extracts only related information from the response.

Updateding Web application/ web page – After processing the response AJAX callback function updates the web page or web application dynamically according to the response. It includes modifying HTML content, displaying error messages, updating the values, etc.

Handle Error – If the request meets an error, then the server may respond with an error status due to any request failure, network issue, etc. So the Handling response process handles the error very efficiently and takes proper action against the error.

How to handle responses works

Follow the following steps to handle the responses using XMLHttpRequest –

Step 1 – Create an XMLHttpRequest object using XMLHttpRequest() constructor. Using this object you can easily do HTTP requests and can handle their responses asynchronously.

```
var qhttp = new XMLHttpRequest();
```

Step 2 – Define an event handler for the readystatechange event. This event trigger whenever the value of the readyState property of the XHR object changes.

```
qhttp.onreadystatechange = function() {
   if (qhttp.readyState == 4){
      if(qhttp.status == 200){
         // Display the response
      }else{
         // Handle the error if occure
      }
    }
  };
```

Step 3 – Open the request by using the HTTP method(like GET, POST, etc) and the URL which we want to request.

```
qhttp.open("HTTP Method","your-URL", true);
```

Step 4 – Set any header if required.

```
qhttp.setRequestHeader('Authorization', 'Your-Token');
```

Step 5 – Send the request to the server.

```
qhttp.send()
```

Example

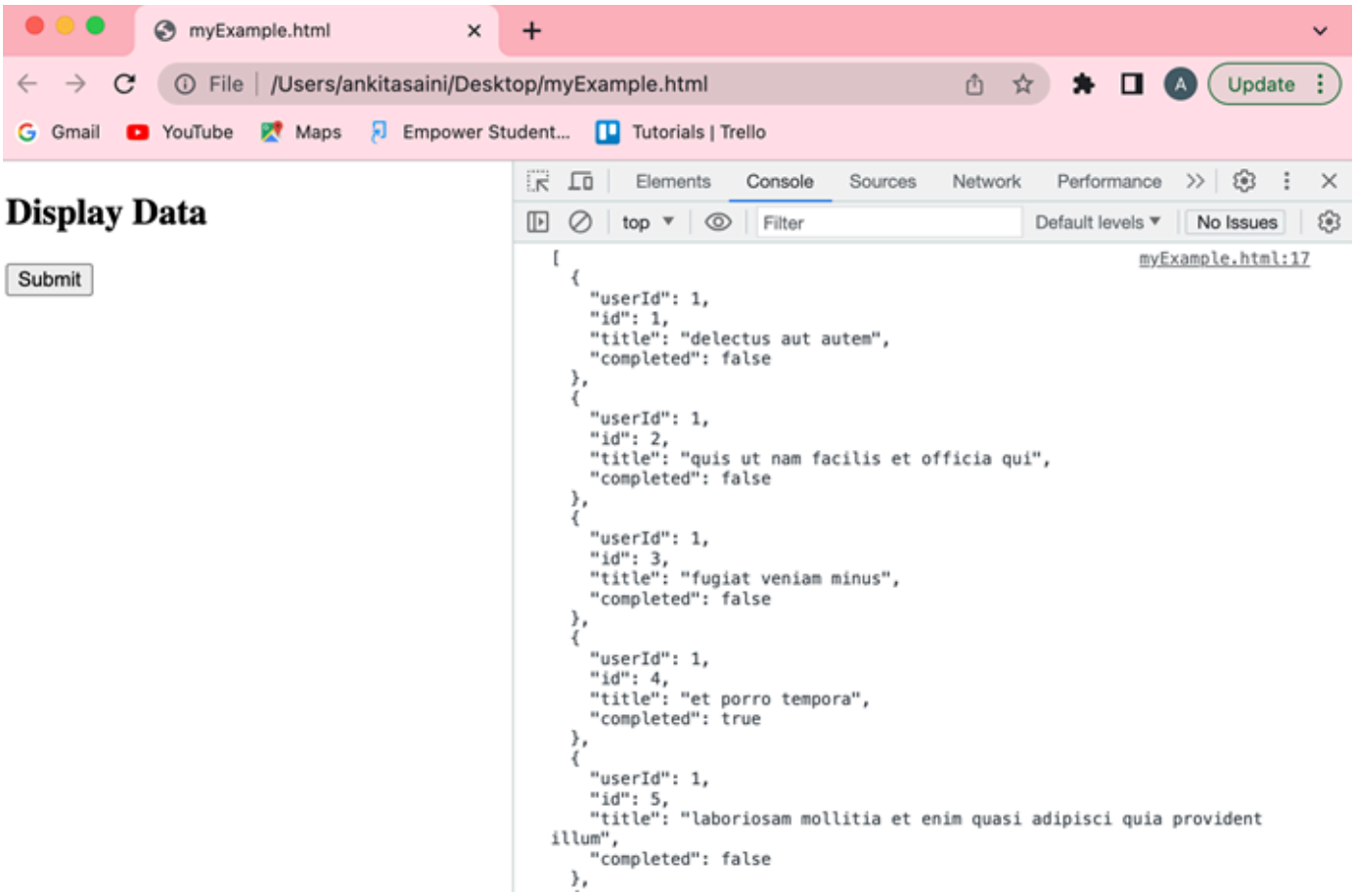
In the following program, we will handle the response returned by the server on the given request. So for that, we will create a Javascript function named as handleResponse() which handles the response returned by the server and display the result accordingly. This function first creates an XMLHttpRequest object, and then it defines an "onreadystatechange" event handler to handle the request state. When the request state changes, then the function checks if the request is complete(readyState = 4). If the request is complete, then the function checks the status code = 200. If the status code is 200, then display the response. Otherwise, display the error message.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   function handleResponse() {
   // Creating XMLHttpRequest object
```

```
    var qhttp = new XMLHttpRequest();
    // Creating call back function
    qhttp.onreadystatechange = function() {
       if (qhttp.readyState == 4){
          if(qhttp.status == 200){
             // Display the response
             console.log(qhttp.responseText)
          }else{
             console.log("Found Error: ", qhttp.status)
          }
       }
    };
    // Open the given file
    qhttp.open("GET", "https://jsonplaceholder.typicode.com/todos", true);
    // Sending request to the server
    qhttp.send()
}
</script>
<h2>Display Data</h2>
<button type="button" onclick="handleResponse()">Submit</button>
<div id="sample"></div>
</body>
</html>
```


Output



Conclusion

So this is how an AJAX can handle the response returned by the server due to which web pages can easily communicate with the server in the background asynchronously without refreshing the whole page. Now in the next article, we will learn how to handle binary data in the AJAX.