BridgeLabz

Employability Delivered

HTML, CSS, Javascript and Ajax Server Calls

BridgeLabz
Employability Delivered

Ajax – Async, Callback & Promise

BridgeLabz
Employability Delivered

UC 1

Demonstrate Async nature of JavaScript

- **Note:** It is similar to Main thread invoking another Activity which are processed by child Thread.
- Run the JS code using Node Compiler

Asynchronous nature of JavaScript (async)

BridgeLabz
Employability Delivered

- Async in JavaScript is about two activities, where in Activity-A triggers another Activity-B, and Activity-B will be completed in future.

```
  GNU nano 2.0.6                          File: timer.js

function showTime() {
   const date = new Date();
   return date.getHours() + "Hrs:"  + date.getMinutes() + "Mins:" +date.getSeconds()+ "Secs";
}

function showSessionExpire(){
  console.log("Activity-B: Your session expired at "+showTime());
}

console.log("Activity-A: Trigerring Activity-B at "+showTime());
setTimeout(showSessionExpire, 5000);
console.log("Activity-A: Triggered Activity-B at "+showTime()+" will execute after 5 seconds");
```

```
NodeDevelopment $ node timer.js
Activity-A: Trigerring Activity-B at 7Hrs:10Mins:16Secs
Activity-A: Triggered Activity-B at 7Hrs:10Mins:16Secs will execute after 5 seconds
Activity-B: Your session expired at 7Hrs:10Mins:21Secs
```

AJAX – Asynchronous JavaScript And XML

BridgeLabz
Employability Delivered

- AJAX is the backbone of any modern web application today and stands for **Asynchronous JavaScript And XML**. Because XML was a standard in 2005.
- In 2015 **JSON** has become the primary data exchange format. therefore XML has nothing to do with the name.
- AJAX is used to communicate between **client and server**.
- Following three lines are always required to send a request to the server.

- **Method Types** – HTTP Methods like GET, POST, PUT & DELETE, etc.
- **URL** – REST Call URL
- **Async** – can be set to true or false.
- **Xhr.Send()** – to make a HTTPRequest to the server

```
var xhr = new XMLHttpRequest();
xhr.open(methodType, URL, async);
xhr.send();
```

BridgeLabz
Employability Delivered

UC 2

Ability to view Employee Data from JSON Server having ID, Name and Salary using AJAX

- Run the JS code using Node Compiler
- Run *npm install xmlhttprequest*

Ajax Demo

BridgeLabz
Employability Delivered

- In order to process the response sent by the server, we need to add callback on the xhr instance when status is in 200 range and readyState is 4.
- This is done by adding listener function to onreadystatechange. This is called whenever there is a change in HTTP Connection. They are
  - **readyState** – values are 0 – 4. when it reaches 3, it means data starts coming from the server and when it reaches 4, it means all the data has come and server has processed the request completely and the connection is closed.
  - **status** :- tells if our request was processed successfully or was there any error. if status isin 200 range, we know our request is processed successfully. Any other code, states that our request has failed.
  - **responseText** – responseText property of xhr instance will hold the value which is returned by the server. It is either in JSON format or XML format.

Ajax Demo and Method Callback

BridgeLabz
Employability Delivered

```
let XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
function makeAJAXCall(methodType, url, callback, async = true, data = null) {
    let xhr = new XMLHttpRequest();
    xhr.onreadystatechange = function(){
        console.log("State Changed Called. Ready State: "+
                    xhr.readyState+" Status:"+xhr.status);
        if (xhr.readyState === 4) {
            // Matching all 200 Series Responses
            if (xhr.status === 200 || xhr.status === 201){
                callback(xhr.responseText);
            } else if (xhr.status >= 400) {
                console.log("Handle 400 Client Error or 500 Server Error");
            }
        }
    }
    xhr.open(methodType, url, async);
    if (data) {
        console.log(JSON.stringify(data));
        xhr.setRequestHeader("Content-Type", "application/json");
        xhr.send(JSON.stringify(data));
    } else xhr.send();
    console.log(methodType+" request sent to the server");
}
```

```
const getURL = "http://127.0.0.1:3000/employees/1";
function getUserDetails(data){
    console.log("Get User Data: "+data)
}
makeAJAXCall("GET", getURL, getUserDetails);

const deleteURL = "http://localhost:3000/employees/4";
function userDeleted(data){
    console.log("User Deleted "+data)
}
makeAJAXCall("DELETE", deleteURL, userDeleted, false);

const postURL = "http://localhost:3000/employees";
const emplData = {"name": "Harry","salary": "5000"};
function userAdded(data){
    console.log("User Added: "+data)
}
makeAJAXCall("POST", postURL, userAdded, true, emplData);
```

More about Callback

BridgeLabz
Employability Delivered

- In the demo, you can see that there are three places where makeAjaxCall function is called and **handling of server response** is different for all the scenarios.
- makeAjaxCall is a kind of **service** function here which takes the ajax details along with callback reference to inform the Caller on completion of service.
- Advantage of Callback –
  - Using the callback reference, we can create a **reusable independent function** which can just focus on making ajax call.
  - In the callback function, we can **process the response data** such as show the user details or listing down the repositories list.
- Disadvantage of Call Back –
  - **Handling of multiple status codes** requires multiple callback functions. For e.g. Success Status codes is 200 Series, Client Error Status Codes is 400 Series and Server Error Status Codes is 500 Series.
  - Here is the list of Status Codes..

BridgeLabz
Employability Delivered

UC 3

Ability to view Employee Data from JSON Server having ID, Name and Salary using AJAX and Promise

- Run the JS code using Node Compiler
- Run *npm install xmlhttprequest*

Ajax Demo and Promise

BridgeLabz
Employability Delivered

- Promise is used to overcome issues with **multiple callbacks** and provide better way to manage **success and error conditions**.
- **Promise is an object** which is returned by the async function like ajax. Promise object has three states
  - **pending** :- means the async operation is going on.
  - **resolved** :- async operation is completed successfully.
  - **rejected** :- async operation is completed with error.

- **Two Parts** to Promise Objects
- Part1 – Inside Async Function
  - Promise object is **created**.
  - Async function returns the **promise object**
  - If async is done successfully, promise object is resolved by calling its **resolve** method.
  - If async is done with error, promise object is rejected by calling its **rejected** method.
- Part2 - Outside Async function
  - Call the function and get the promise object
  - Attach success handler, error handler on the promise object using then method

Ajax Demo and Promise

BridgeLabz
Employability Delivered

```
let XMLHttpRequest = require("xmlhttprequest").XMLHttpRequest;
function makePromiseCall(methodType, url, async = true, data = null) {
    return new Promise(function (resolve, reject) {
        let xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function(){
            console.log("State Changed Called. Ready State: "+
                        xhr.readyState+" Status:"+xhr.status);
            if (xhr.status.toString().match('^[2][0-9]{2}$')) {
                resolve(xhr.responseText);
            } else if (xhr.status.toString().match('^[4,5][0-9]{2}$')) {
                reject({
                    status: xhr.status,
                    statusText: xhr.statusText
                });
                console.log("XHR Failed");
            }
        }
        xhr.open(methodType, url, async);
        if (data) {
            console.log(JSON.stringify(data));
            xhr.setRequestHeader("Content-Type", "application/json");
            xhr.send(JSON.stringify(data));
        } else xhr.send();
        console.log(methodType+" request sent to the server");
    });
}
```

```
const getURL = "http://127.0.0.1:3000/employees/1";
makePromiseCall("GET", getURL, true)
    .then(responseText => {
        console.log("Get User Data: "+responseText)
    })
    .catch(error => console.log("GET Error Status: "+
                                JSON.stringify(error)));

const deleteURL = "http://localhost:3000/employees/4";
makePromiseCall("DELETE", deleteURL, false)
    .then(responseText => {
        console.log("User Deleted: "+responseText)
    })
    .catch(error => console.log("DELETE Error Status: "+
                                JSON.stringify(error)));

const postURL = "http://localhost:3000/employees";
const emplData = {"name": "Harry","salary": "5000"};
makePromiseCall("POST", postURL, true, emplData)
    .then(responseText => {
        console.log("User Added: "+responseText)
    })
    .catch(error => console.log("POST Error Status: "+
                                JSON.stringify(error)));
```

BridgeLabz
Employability Delivered

UC 4

Create a simple html file to test the REST services with JSON Server using Promise

- Create http_services.js file and copy the Ajax Promise Function to it.
- **Note** – Instead of setting onreadystatechange listener we will set onload and onerror listener
- Call the Promise function from the test services html and populate the result in the html div element
- EmployeeDB.Json File

Test Services HTML

BridgeLabz
Employability Delivered

The Test Serices Result

Get User Data: { "id": 1, "name": "Pankaj", "salary": "10000" }

Delete Data: {}

New User Data: { "name": "Harry", "salary": "5000", "id": 4 }

Test Services HTML


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
  <title>Test Ajax Services and Promise</title>
  <style>
    .body { margin: auto; }
    div {
      border: 1px solid black;
      background-color: lightblue;
      padding: 50px;
      width: 80%;
      font-size: xx-large;
      margin-top: 50px;
    }
  </style>
  <script src="../services/http_services.js"></script>
</head>
```

```
<body>
  <h1>The Test Serices Result</h1>
  <div id="get_services" ></div>
  <div id="delete_services" ></div>
  <div id="post_services" ></div>
  <script>
    const getElem = document.querySelector("#get_services");
    const getURL = "http://127.0.0.1:3000/employees/1";
    makePromiseCall("GET", getURL, true)
      .then(responseText => {
          getElem.textContent = "Get User Data: "+responseText;
      })
      .catch(error => {
        getElem.textContent = "GET Error Status: "+JSON.stringify(error);
      });

    const deleteElem = document.querySelector("#delete_services");
    const deleteURL = "http://127.0.0.1:3000/employees/4";
    makePromiseCall("DELETE", deleteURL, false)
      .then(responseText => {
        deleteElem.textContent = "Delete Data: "+responseText;
      })
      .catch(error => {
        deleteElem.textContent = "DELETE Error Status: "+JSON.stringify(error);
      });

    const postElem = document.querySelector("#post_services");
    const postURL = "http://127.0.0.1:3000/employees";
    const emplData = {"name": "Harry","salary": "5000"};
    makePromiseCall("POST", postURL, true, emplData)
      .then(responseText => {
        postElem.textContent = "New User Data: "+responseText;
      })
      .catch(error => {
        postElem.textContent = "POST Error Status: "+JSON.stringify(error);
      });
  </script>
</body>
</html>
```

```
{
  "employees": [
    {
      "id": 1,
      "name": "Pankaj",
      "salary": "10000"
    },
    {
      "id": 2,
      "name": "Lisa",
      "salary": "8000"
    },
    {
      "name": "Terisa",
      "salary": "10000",
      "id": 3
    },
    {
      "name": "Harry",
      "salary": "5000",
      "id": 4
    }
  ]
}
```

Http Services implementing Ajax Promise Function

services > JS http_services.js > ...

```
function makePromiseCall(methodType, url, async = true, data = null) {
    return new Promise(function (resolve, reject) {
        let xhr = new XMLHttpRequest();
        xhr.onload = function(){
            console.log(methodType+" State Changed Called. Ready State: "+
                        xhr.readyState+" Status:"+xhr.status);
            if (xhr.status.toString().match('^[2][0-9]{2}$')) {
                resolve(xhr.responseText);
            } else if (xhr.status.toString().match('^[4,5][0-9]{2}$')) {
                reject({
                    status: xhr.status,
                    statusText: xhr.statusText
                });
                console.log("XHR Failed");
            }
        }
        xhr.onerror = function () {
          reject({
                status: this.status,
                statusText: xhttp.statusText
          });
        };
        xhr.open(methodType, url, async);
        if (data) {
            console.log(JSON.stringify(data));
            xhr.setRequestHeader("Content-Type", "application/json");
            xhr.send(JSON.stringify(data));
        } else xhr.send();
        console.log(methodType+" request sent to the server");
    });
}
```

BridgeLabz
Employability Delivered

References

BridgeLabz
Employability Delivered

- Ajax, Async, Callback and Promise
- JS Promises and Async
- All HTTP Status Codes

BridgeLabz

Employability Delivered

Thank You