🌐 Complete AJAX, JSON, Cookies & Error Handling Guide

I'll explain **AJAX**, **XMLHttpRequest**, **JSON**, **Cookies**, **Date Object**, and **Error Handling** in detail.

Part 1: AJAX - Asynchronous JavaScript and XML

What is AJAX?

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML

W3Schools Reference: AJAX allows web pages to update asynchronously by exchanging data with a server behind the scenes.

Key Points:

- Update page **without reloading**
- Request data from server **after page loads**
- Receive data from server **asynchronously**
- Send data to server **in background**

Part 2: XMLHttpRequest Object

Basic AJAX Request

```
var xhr = new XMLHttpRequest()
xhr.open("GET", "https://jsonplaceholder.typicode.com/users/8")
xhr.send()

xhr.onreadystatechange = function(){
    if(xhr.readyState == 4 && xhr.status == 200){
        console.log(xhr.response)
        var result = JSON.parse(xhr.response)
        console.log(result)
    }
}
```

Step-by-Step Breakdown:

1. Create XMLHttpRequest Object

javascript

```
var xhr = new XMLHttpRequest()
```

2. Open Connection

javascript

```
xhr.open(method, url, async)
```

Parameters:

- `method` : "GET", "POST", "PUT", "DELETE"
- `url` : API endpoint
- `async` : true (asynchronous) or false (synchronous)

Examples:

javascript

```
xhr.open("GET", "https://api.example.com/users")
xhr.open("POST", "https://api.example.com/users", true)
xhr.open("DELETE", "https://api.example.com/users/5")
```

3. Send Request

javascript

```
xhr.send()         // For GET
xhr.send(data)     // For POST/PUT
```

Ready State Values

```
0 - UNSENT          Request not initialized
1 - OPENED          Connection established (open() called)
```

```
2 - HEADERS_RECEIVED Server received request
3 - LOADING          Processing request (downloading data)
4 - DONE             Request finished, response ready
```

Tracking Ready State:

javascript

```
xhr.onreadystatechange = function(){
    console.log("Ready State:", xhr.readyState)

    if(xhr.readyState == 0) console.log("Not initialized")
    if(xhr.readyState == 1) console.log("Connection opened")
    if(xhr.readyState == 2) console.log("Headers received")
    if(xhr.readyState == 3) console.log("Loading...")
    if(xhr.readyState == 4) console.log("Done!")
}
```

HTTP Status Codes

javascript

```
if(xhr.status == 200) console.log("OK - Success")
if(xhr.status == 201) console.log("Created")
if(xhr.status == 400) console.log("Bad Request")
if(xhr.status == 404) console.log("Not Found")
if(xhr.status == 500) console.log("Server Error")
```

Common Status Codes:

- **200** - OK (Success)
- **201** - Created
- **204** - No Content
- **400** - Bad Request
- **401** - Unauthorized
- **403** - Forbidden
- **404** - Not Found
- **500** - Internal Server Error

Complete AJAX Example with Table Display

HTML:

html

```
<table>
  <thead>
    <th>Id</th>
    <th>Name</th>
    <th>Email</th>
  </thead>
  <tbody id="result"></tbody>
</table>
```

JavaScript:

javascript

```
var xhr = new XMLHttpRequest()
xhr.open("GET", "https://jsonplaceholder.typicode.com/users/8")
xhr.send()

xhr.onreadystatechange = function(){
    if(xhr.readyState == 4 && xhr.status == 200){
        // Parse JSON response
        var result = JSON.parse(xhr.response)
        console.log(result)

        // Display in table
        var tbody = document.getElementById("result")
        tbody.innerHTML = `
            <tr>
                <td>${result.id}</td>
                <td>${result.name}</td>
                <td>${result.email}</td>
            </tr>
        `
    }
}
```

Reusable AJAX Function

javascript

```javascript
function getUserId(id){
    var xhr = new XMLHttpRequest()
    xhr.open("GET", `https://jsonplaceholder.typicode.com/users/${id}`)
    xhr.send()

    xhr.onreadystatechange = function(){
        if(xhr.readyState == 4 && xhr.status == 200){
            var result = JSON.parse(xhr.response)
            console.log(result)
            getUser(result)
        }
    }
}

function getUser(user){
    var tbody = document.getElementById("result")
    tbody.innerHTML = `
        <tr>
            <td>${user.id}</td>
            <td>${user.email}</td>
            <td>${user.name}</td>
        </tr>
    `
}

// Usage
getUserId(3)  // Gets user with ID 3
getUserId(7)  // Gets user with ID 7
```

Part 3: XML vs JSON

XML - eXtensible Markup Language

xml

```xml
<product>
    <id>1</id>
    <name>p1</name>
    <price>100</price>
</product>
```

Characteristics:

- Uses tags like HTML
- More verbose (larger file size)
- Harder to parse
- Less common now

JSON - JavaScript Object Notation ✅

json

```
{
    "id": 1,
    "name": "p1",
    "price": 100
}
```

Characteristics:

- Lightweight and readable
- Easy to parse in JavaScript
- **Most commonly used** today
- Native JavaScript support

Working with JSON

JSON.parse() - Convert JSON string to Object

javascript

```
var jsonString = '{"id":1,"name":"Ahmed"}'
var obj = JSON.parse(jsonString)

console.log(obj.id)    // 1
console.log(obj.name)  // Ahmed
```

JSON.stringify() - Convert Object to JSON string

javascript

```javascript
var obj = {
    id: 1,
    name: "Ahmed",
    email: "ahmed@gmail.com"
}

var jsonString = JSON.stringify(obj)
console.log(jsonString)
// {"id":1,"name":"Ahmed","email":"ahmed@gmail.com"}
```

Scenarios:

javascript

```javascript
// Parse API response
var xhr = new XMLHttpRequest()
xhr.onreadystatechange = function(){
    if(xhr.readyState == 4 && xhr.status == 200){
        var data = JSON.parse(xhr.response)
        console.log(data)
    }
}

// Send data to server
var userData = {name: "Ahmed", age: 25}
var jsonData = JSON.stringify(userData)
xhr.send(jsonData)

// Store array in JSON
var colors = ["red", "green", "blue"]
var jsonColors = JSON.stringify(colors)
console.log(jsonColors)  // ["red","green","blue"]
```

Part 4: Date Object

javascript

```javascript
var date = new Date()
```

W3Schools Reference: Date object works with dates and times.

Getting Date Components

javascript

```
console.log(date.getDate())        // Day of month (1-31)
console.log(date.getDay())         // Day of week (0-6, 0=Sunday)
console.log(date.getMonth())       // Month (0-11, 0=January)
console.log(date.getFullYear())    // Year (2024)
console.log(date.getHours())       // Hour (0-23)
console.log(date.getMinutes())     // Minutes (0-59)
console.log(date.getSeconds())     // Seconds (0-59)
console.log(date.getTime())        // Milliseconds since 1970
```

Setting Date Components

javascript

```
// Add 30 days to current date
date.setDate(date.getDate() + 30)
console.log(date)

// Set specific date
date.setFullYear(2025)
date.setMonth(5)      // June (0-indexed)
date.setDate(15)      // 15th day

// Subtract 3 days
date.setDate(date.getDate() - 3)
```

Scenarios:

javascript

```
// Get tomorrow's date
var tomorrow = new Date()
tomorrow.setDate(tomorrow.getDate() + 1)

// Get date 7 days from now
var nextWeek = new Date()
nextWeek.setDate(nextWeek.getDate() + 7)

// Get last day of month
var lastDay = new Date(2024, 12, 0)  // Month 12, day 0 = last day of Nov
```

```javascript
// Compare dates
var date1 = new Date("2024-01-01")
var date2 = new Date("2024-12-31")
if(date1 < date2){
    console.log("date1 is earlier")
}

// Format date
var now = new Date()
console.log(now.toLocaleDateString())  // 12/15/2024
console.log(now.toLocaleTimeString())  // 2:30:45 PM
console.log(now.toLocaleString())      // 12/15/2024, 2:30:45 PM
```

Part 5: Cookies

W3Schools Reference: Cookies are data stored in small text files on user's computer.

Types of Cookies:

1. Session Cookie

- Deleted when browser closes
- No expiration date
- Temporary

2. Persistent Cookie

- Stored until expiration date
- Survives browser restart
- Long-term storage

Setting Cookies

Basic Cookie

javascript

```javascript
document.cookie = "name=ahmed"
```

Cookie with Expiration

javascript

```javascript
var date = new Date()
date.setDate(date.getDate() + 30)  // Expires in 30 days

document.cookie = `email=ahmed@gmail.com; expires=${date}`
```

Cookie with Path

javascript

```javascript
document.cookie = "username=ahmed; path=/"
```

Reusable Cookie Functions

Save Cookie

javascript

```javascript
function saveCookie(key, value){
    if(typeof value === "object"){
        // Convert object/array to JSON string
        document.cookie = `${key}=${JSON.stringify(value)}`
    }
    else{
        document.cookie = `${key}=${value}`
    }
}
```

Usage:

javascript

```javascript
// Simple cookie
saveCookie("name", "Ahmed")

// Array cookie
saveCookie("colors", ["red", "green", "blue"])

// Object cookie
saveCookie("user", {id: 1, name: "Sara", age: 25})
```

Remove Cookie

javascript

```
function removeCookie(key){
    var date = new Date()
    date.setDate(date.getDate() - 1)  // Set to past date
    document.cookie = `${key}=; expires=${date}`
}
```

How it works: Setting expiration to past date deletes cookie

Usage:

javascript

```
removeCookie("name")
removeCookie("email")
```

Get Cookie

javascript

```
function getCookie(key){
    var cookies = document.cookie.split("; ")

    for(var i = 0; i < cookies.length; i++){
        var cookie = cookies[i].split("=")
        if(cookie[0] === key){
            return cookie[1]
        }
    }
    return null
}
```

Usage:

javascript

```
var userName = getCookie("name")
console.log(userName)  // "Ahmed"

var colors = JSON.parse(getCookie("colors"))
console.log(colors)    // ["red", "green", "blue"]
```

Complete Cookie Management System

javascript

```
// Cookie Manager
var CookieManager = {
    // Set cookie
    set: function(key, value, days){
        var expires = ""
        if(days){
            var date = new Date()
            date.setDate(date.getDate() + days)
            expires = `; expires=${date.toUTCString()}`
        }

        var val = typeof value === "object" ? JSON.stringify(value) : value
        document.cookie = `${key}=${val}${expires}; path=/`
    },

    // Get cookie
    get: function(key){
        var name = key + "="
        var cookies = document.cookie.split("; ")

        for(var i = 0; i < cookies.length; i++){
            var cookie = cookies[i]
            if(cookie.indexOf(name) === 0){
                var value = cookie.substring(name.length)
                try{
                    return JSON.parse(value)
                } catch(e){
                    return value
                }
            }
        }
        return null
    },
```

```
    // Remove cookie
    remove: function(key){
        document.cookie = `${key}=; expires=Thu, 01 Jan 1970 00:00:00 UTC;
path=/`
    },

    // Check if cookie exists
    exists: function(key){
        return this.get(key) !== null
    },

    // Get all cookies
    getAll: function(){
        var cookies = {}
        var all = document.cookie.split("; ")

        for(var i = 0; i < all.length; i++){
            var cookie = all[i].split("=")
            cookies[cookie[0]] = cookie[1]
        }
        return cookies
    }
}

// Usage
CookieManager.set("username", "Ahmed", 7)        // Expires in 7 days
CookieManager.set("settings", {theme: "dark"}, 30)
console.log(CookieManager.get("username"))       // "Ahmed"
console.log(CookieManager.exists("username"))    // true
CookieManager.remove("username")
console.log(CookieManager.getAll())              // All cookies
```

Part 6: Error Handling - try...catch...finally

W3Schools Reference: try...catch statement handles errors without stopping script execution.

Basic Syntax

javascript

```
try{
    // Code that might throw error
```

```
    console.log("Hello PD")
    console.logg("Error here")  // ❌ Typo: logg instead of log
    console.log("This won't execute")
}
catch(error){
    // Handle error
    console.log("Error occurred:", error)
}
finally{
    // Always executes (optional)
    console.log("Finally block")
}
```

Output:
```
Hello PD
Error occurred: ReferenceError: logg is not defined
Finally block
```

Error Object Properties

javascript

```
try{
    undefinedFunction()
}
catch(error){
    console.log(error.name)        // "ReferenceError"
    console.log(error.message)     // "undefinedFunction is not defined"
    console.log(error.stack)       // Stack trace
}
```

Throwing Custom Errors

javascript

```
function divide(a, b){
    if(b === 0){
        throw new Error("Cannot divide by zero")
```

```
    }
    return a / b
}

try{
    var result = divide(10, 0)
}
catch(error){
    console.log("Error:", error.message)
}
```

Different Error Types

javascript

```
// ReferenceError
try{
    console.log(nonExistentVariable)
}
catch(error){
    console.log(error.name)  // "ReferenceError"
}

// TypeError
try{
    var num = 5
    num.toUpperCase()  // Numbers don't have toUpperCase()
}
catch(error){
    console.log(error.name)  // "TypeError"
}

// SyntaxError (caught at compile time)
try{
    eval("var x = ;")  // Invalid syntax
}
catch(error){
    console.log(error.name)  // "SyntaxError"
}

// RangeError
try{
    var arr = new Array(-1)  // Negative array length
```

```
}
catch(error){
    console.log(error.name)  // "RangeError"
}
```

Practical Examples

1. AJAX Error Handling

javascript

```
function getUserData(id){
    var xhr = new XMLHttpRequest()
    xhr.open("GET", `https://api.example.com/users/${id}`)
    xhr.send()

    xhr.onreadystatechange = function(){
        if(xhr.readyState == 4){
            try{
                if(xhr.status == 200){
                    var data = JSON.parse(xhr.response)
                    displayUser(data)
                }
                else{
                    throw new Error(`HTTP Error: ${xhr.status}`)
                }
            }
            catch(error){
                console.error("Error:", error.message)
                alert("Failed to load user data")
            }
        }
    }
}
```

---

2. JSON Parse Error Handling

javascript

```
function parseJSON(jsonString){
    try{
        var obj = JSON.parse(jsonString)
        return obj
    }
    catch(error){
        console.error("Invalid JSON:", error.message)
        return null
    }
}

var result = parseJSON("{invalid json}")
if(result === null){
    console.log("Parsing failed")
}
```

---

3. Form Validation

javascript

```
function validateForm(data){
    try{
        if(!data.name){
            throw new Error("Name is required")
        }
        if(!data.email || !data.email.includes("@")){
            throw new Error("Valid email is required")
        }
        if(data.age < 18){
            throw new Error("Must be 18 or older")
        }

        console.log("Validation passed")
        return true
    }
    catch(error){
        alert(error.message)
        return false
    }
}

var formData = {
    name: "",
```

```
    email: "invalid",
    age: 15
}

validateForm(formData)
```

---

4. Finally Block Usage

javascript

```
var file = null

try{
    file = openFile("data.txt")
    processFile(file)
}
catch(error){
    console.error("Error processing file:", error.message)
}
finally{
    // Always close file, even if error occurred
    if(file){
        closeFile(file)
    }
    console.log("Cleanup completed")
}
```

Part 7: Function Arguments Validation

javascript

```
function sum(x, y){
    if(arguments.length < 2){
        console.log("You must pass 2 parameters")
    }
    else if(arguments.length > 2){
        console.log("You must pass only 2 parameters")
    }
    else{
        console.log(x + y)
```

```
    }
}

sum(1, 3)      // 4
sum(1)         // "You must pass 2 parameters"
sum(1, 2, 3)   // "You must pass only 2 parameters"
```

Better with try...catch:

javascript

```
function sum(x, y){
    try{
        if(arguments.length !== 2){
            throw new Error(`Expected 2 arguments, got ${arguments.length}`)
        }
        if(typeof x !== "number" || typeof y !== "number"){
            throw new Error("Both arguments must be numbers")
        }

        return x + y
    }
    catch(error){
        console.error(error.message)
        return null
    }
}

console.log(sum(5, 10))       // 15
console.log(sum(5))           // Error: Expected 2 arguments, got 1
console.log(sum("5", 10))     // Error: Both arguments must be numbers
```

📊 Summary Tables

AJAX Ready States

State	Value	Meaning
UNSENT	0	Not initialized
OPENED	1	Connection opened
HEADERS_RECEIVED	2	Request received

State	Value	Meaning
LOADING	3	Processing request
DONE	4	Request complete

HTTP Status Codes

Code	Meaning
200	OK - Success
201	Created
400	Bad Request
401	Unauthorized
404	Not Found
500	Server Error

Error Types

Type	Description
ReferenceError	Variable not found
TypeError	Wrong type used
SyntaxError	Invalid syntax
RangeError	Value out of range

🎯 Best Practices:

1. **Always check readyState AND status** in AJAX

javascript

```
    if(xhr.readyState == 4 && xhr.status == 200)
```

2. **Use JSON.parse() inside try...catch**

javascript

```
try{
    var data = JSON.parse(response)
} catch(e){
    console.error("Invalid JSON")
}
```

3. **Set cookie expiration explicitly**

javascript

```
var date = new Date()
date.setDate(date.getDate() + 7)  // 7 days
document.cookie = `key=value; expires=${date}`
```

4. **Always use finally for cleanup**

javascript

```
finally{
    // Close connections, clear resources
}
```

5. **Validate function arguments**

javascript

```
if(arguments.length !== expectedCount){
    throw new Error("Invalid arguments")
}
```

BY. Abdullah Ali

Contact : +201012613453