AJAX Lab Manual

Introduction to AJAX

What is AJAX?

AJAX (Asynchronous JavaScript and XML) is a web development technique that allows web pages to be updated asynchronously by exchanging data with a server behind the scenes. This means updates can happen without reloading the entire page.

Key Benefits

- Updates page content without refresh
- Improves user experience
- Reduces server load
- Enables real-time data updates
- Better interactivity

AJAX Fundamentals

The XMLHttpRequest Object

The core of AJAX functionality. Used to exchange data with a server.

Basic Structure

```
// Create XMLHttpRequest object
const xhr = new XMLHttpRequest();

// Configure request
xhr.open('GET', 'url-here', true);

// Set up response handling
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
    // Handle response
    console.log(xhr.responseText);
    }
};

// Send request
xhr.send();
```

ReadyState Values

- 0: UNSENT - Client created, open() not called
- 1: OPENED - open() called
- 2: HEADERS_RECEIVED - send() called, headers received
- 3: LOADING - Downloading data
- 4: DONE - Operation complete

Common HTTP Status Codes

- 200: OK
- 201: Created
- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 500: Internal Server Error

Implementation Methods

Method 1: XMLHttpRequest

```
function makeRequest(url, method = 'GET') {
    return new Promise((resolve, reject) => {
        const xhr = new XMLHttpRequest();
        xhr.open(method, url, true);

        xhr.onload = function() {
            if (xhr.status >= 200 && xhr.status < 300) {
                resolve(xhr.response);
            } else {
                reject({
                    status: xhr.status,
                    statusText: xhr.statusText
                });
            }
        };

        xhr.onerror = function() {
            reject({
                status: xhr.status,
                statusText: xhr.statusText
            });
        };

        xhr.send();
    });
}
```

Method 2: jQuery AJAX

```
// Simple GET request
$.get('url-here', function(data) {
    console.log(data);
});

// Full AJAX request
$.ajax({
    url: 'url-here',
    method: 'GET',
    dataType: 'json',
    success: function(response) {
        console.log(response);
    },
    error: function(xhr, status, error) {
        console.error(error);
    }
});
```

Method 3: Fetch API (Modern Approach)

```
fetch('url-here')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error:', error));

// With more options
fetch('url-here', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(data)
})
    .then(response => response.json())
    .then(data => console.log(data));
```

Working with APIs

Making API Requests

```
$.ajax({
    url: 'https://api.example.com/data',
    method: 'GET',
    headers: {
        'Authorization': 'Bearer token-here'
    },
    success: function(response) {
        // Handle success
    },
    error: function(xhr, status, error) {
        // Handle error
    }
});
```

Handling JSON Data

```
// Parsing JSON
const jsonString = '{"name": "John", "age": 30}';
const data = JSON.parse(jsonString);
```

```
// Stringifying JSON
const obj = {name: "John", age: 30};
const jsonStr = JSON.stringify(obj);
```

Error Handling

Try-Catch Blocks

```
try {
    // AJAX operation here
} catch (error) {
    console.error('Error:', error);
    // Handle error appropriately
}
```

Promise Error Handling

```
fetch('url-here')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .catch(error => {
        console.error('Error:', error);
        // Show user-friendly error message
    });
```