AJAX & FETCH API

TechRacine Academy

https://techracine.com

https://academy.techracine.com

Frontend Developer Intermediate Path

Table of Contents

1. Introduction to AJAX

AJAX stands for Asynchronous JavaScript and XML.

AJAX allows webpages to communicate with servers without reloading the page.

It improves responsiveness and user experience.

Modern SaaS applications and dashboards heavily depend on AJAX techniques.

Today, JSON is commonly used instead of XML in AJAX applications.

2. Why AJAX is Important

AJAX enables real-time interactions in web applications.

Users can load data dynamically without refreshing pages.

AJAX improves performance and usability.

Applications such as Gmail, dashboards, chat systems, and admin portals rely heavily on AJAX.

3. Traditional Request vs AJAX

Traditional requests reload the entire webpage.

AJAX only updates required sections dynamically.

This significantly improves user experience.

4. Understanding HTTP Requests

AJAX communication depends on HTTP requests.

Common request methods include GET, POST, PUT, and DELETE.

Understanding HTTP is essential for API integration.

```
GET /api/users
POST /api/save
```

5. AJAX using XMLHttpRequest

XMLHttpRequest was the original AJAX implementation in JavaScript.

It allows communication with backend servers.

```
const xhr = new XMLHttpRequest();

xhr.open("GET", "data.json");

xhr.onload = function() {

    console.log(xhr.responseText);

};

xhr.send();
```

6. jQuery AJAX Basics

jQuery simplified AJAX significantly.

Developers could perform server communication with minimal code.

```
$.ajax({

    url: "data.php",
    method: "GET",

    success: function(data) {

        console.log(data);

    }

});
```

7. Working with JSON Data

JSON stands for JavaScript Object Notation.

Most APIs today use JSON for data exchange.

JSON is lightweight and easy to parse.

```
const user = {

    name: "John",
    role: "Developer"

};
```

8. GET and POST Requests

GET requests retrieve data from servers.

POST requests send data to servers.

Understanding request methods is critical for API integration.

```
fetch("/api/users")

fetch("/api/save", {
    method: "POST"
});
```

9. Error Handling in AJAX

Error handling improves reliability.

Applications should properly handle server failures and invalid responses.

```
$.ajax({

    error: function(error) {

        console.log(error);

    }

});
```

10. Introduction to Fetch API

Fetch API is the modern replacement for XMLHttpRequest.

It provides cleaner and promise-based syntax.

Modern frontend applications heavily depend on Fetch.

```
fetch("data.json")

.then(response => response.json())

.then(data => console.log(data));
```

11. Fetch GET Requests

GET requests retrieve data from APIs.

Fetch supports asynchronous communication.

```
fetch("/api/users")

.then(response => response.json())

.then(data => {

    console.log(data);

});
```

12. Fetch POST Requests

POST requests send data to APIs.

Applications commonly use POST for forms and submissions.

```
fetch("/api/save", {

    method: "POST",

    headers: {
        "Content-Type": "application/json"
    },

    body: JSON.stringify({
        name: "John"
    })

});
```

13. Async and Await with Fetch

Async and await simplify asynchronous programming.

They improve readability compared to promise chaining.

```
async function loadUsers() {

    const response =
    await fetch("/api/users");

    const data =
    await response.json();

    console.log(data);

}
```

14. Handling API Responses

API responses should be validated before usage.

Applications should check status codes and response structures.

```
if(response.ok) {

    console.log("Success");

}
```

15. Working with REST APIs

REST APIs are widely used in modern software development.

Frontend applications communicate with backend systems using APIs.

Understanding REST principles is important for full-stack development.

16. Loading Dynamic Content

AJAX enables dynamic page updates.

Dashboards commonly use dynamic content loading.

```
$("#content").load("menu.html");
```

17. Authentication and Tokens

Modern APIs often require authentication tokens.

JWT tokens are commonly used for secure API access.

```
headers: {
    Authorization: "Bearer TOKEN"
}
```

18. Common API Status Codes

HTTP status codes indicate request results.

200 means success, 404 means not found, and 500 means server error.

Developers should understand status codes during debugging.

19. Performance Considerations

API calls should be optimized to reduce delays.

Applications should avoid unnecessary requests.

Caching improves performance.

20. Security Best Practices

Validate API inputs properly.

Do not expose sensitive tokens in frontend code.

Use HTTPS for API communication.

Handle authentication securely.

21. Real World Usage

AJAX and Fetch API are used in SaaS applications, AI dashboards, enterprise portals, eCommerce systems, and ERP software.

Modern frontend engineering heavily depends on API communication.

22. Summary

AJAX and Fetch API are essential technologies in modern web development.

Understanding asynchronous communication, JSON, REST APIs, and Fetch improves frontend engineering skills.

Modern applications rely heavily on API-driven architectures.

23. Exercises

1. Fetch data from a sample API.
2. Create a dynamic user list.
3. Practice POST requests.
4. Build AJAX-based form submission.
5. Handle API errors properly.
6. Create a live search feature.
7. Practice async-await syntax.
8. Build a weather API integration.
9. Create dynamic dashboard widgets.
10. Practice API authentication headers.