AJAX

CS-493: Enterprise Application Development

Supervisor
Mr. Atif Hussain

Submitted by:

Amad Irfan 2021-CS-25

University of Engineering and Technology
Lahore, Pakistan

Contents

1 AJAX

1.1 Introduction

Asynchronous JavaScript and XML (Ajax) refer to a group of technologies that are used to develop web applications. By combining these technologies, web pages appear more responsive since small packets of data are exchanged with the server and web pages are not reloaded each time that a user makes an input change AJAX is not a programming language. AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. AJAX is a combination of:

- Built-in XMLHttpRequest.
- JavaScript and HTML DOM.

Ajax incorporates these technologies to create a new approach to developing web applications. Ajax defines a method of initiating client to server communication without page reloads. It provides a way to enable partial page updates. From a web page user perspective, it means improved interaction with a web application, which gives the user more control of their environment, similar to that of a desktop application.
In a traditional web application, HTTP requests, that are initiated by the user's interaction with the web interface, are made to a web server. The web server processes the request and returns an HTML page to the client. During HTTP transport, the user is unable to interact with the web application.

1.2 Ajax limitations:

- **Browser support**: Not all browsers support JavaScript or XMLHttpRequest objects. Even among browsers that do have support for JavaScript and XMLHttpRequest, these objects can be treated differently. Each browser's implementation of Ajax must be considered.
- **Security and user privacy** : Not all concerns are addressed. Issues surrounding security and user privacy need to be considered when developing an Ajax application.
- **Bookmark and navigation** : Since Ajax is used to asynchronously load bits of content into an existing page, some of the page information may not correspond to a newly loaded page. Browser history and bookmarks may not have the correct behavior since the URL was unchanged despite parts of the page being changed.

2 How Does AJAX Work?

- **XHTML and CSS** – for presenting the information.
- **The Document Object Model (DOM)** – for the dynamic display of data and its interaction.
- **XML, HTML, and XSLT** – for data interchange and manipulation. However, many developers have replaced XML with JSON since it originated from JavaScript.
- **XMLHttpRequest object** – allows asynchronous communication with the web server.

- **JavaScript** – the programming language that links all these web technologies.

The general principles of AJAX are simple. However, having existing technical knowledge will help you understand the workflow faster.

3 Comparision between Conventional Model AJAX Model

Conventional Model	**AJAX Model**
The browser sends an HTTP request to the server.	The browser creates a JavaScript call, which then creates a new XMLHttpRequest object.
The web server receives and processes the request.	The new XMLHttpRequest object transfers data between the browser and the web server in an XML format.
The web server sends the requested data to the browser.	The XMLHttpRequest object sends a request for the updated page data to the web server. Subsequently, the latter processes the request and sends it back to the browser.
The browser receives the data from the server and reloads it as an HTML page. Users have to wait until it finishes loading. Therefore, the conventional model increases the load on the server and is more time-consuming.	The browser uses JavaScript to process the response and displays the updated content directly on the HTML page without reloading.

Table 1: Comparison between Conventional and AJAX Models

3.1 Role of XMLHttpRequest in AJAX

The XMLHttpRequest (XHR) object plays a crucial role in the asynchronous communication that AJAX relies on. the XHR object became a standard feature across all major browsers. Its primary purpose is to enable the exchange of data between a web browser and a web server asynchronously, without requiring a full page reload. Here's a brief overview of the role of XMLHttpRequest in AJAX:

- **Initiating Requests:** With the XHR object, JavaScript in a web page can create an HTTP request to a server, specifying the type of request (GET or POST), the URL, and other parameters.
- **Asynchronous Communication:** The term "asynchronous" in AJAX comes from the ability of the XHR object to send requests to the server and receive responses in the background without blocking the main execution thread of the web page. This allows other tasks to continue while waiting for the server response.
- **Handling Responses:** The XHR object can be configured to handle different types of server responses, such as plain text, XML, or JSON. Upon receiving a response, JavaScript code can then process the data and update the DOM dynamically.

As AJAX has evolved, there has been a notable shift from using XML as the data format to JSON (JavaScript Object Notation). JSON is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON has several advantages over XML in the context of AJAX:

Simplicity and Readability: JSON has a simpler syntax than XML, making it easier to read and write. This simplicity results in reduced data size and improved efficiency.

Native JavaScript Support: JSON is a natural fit for JavaScript because its syntax closely resembles JavaScript object literal notation. This native support simplifies data manipulation in the client-side code.

Faster Parsing: JSON parsing is typically faster than XML parsing, leading to improved performance in processing data on the client side.

Less Overhead: JSON does not require as much markup as XML, resulting in less overhead in terms of data siz

3.2 Limitations

- **Browser support**
  Not all browsers can support JavaScript and XMLHttpRequest objects. These objects can be handled differently by browsers with JavaScript or XMLHttpRequest support. Ajax implementations in different browsers must be taken into consideration.

- **Security**
  Not all concerns can be addressed. When developing Ajax applications, security and privacy issues must be considered.

- **Accessibility**
  Not all browsers support JavaScript or XMLHttpRequest objects, so you must ensure that your web application is accessible to all users.

- **Bookmark and navigation**
  Search engine Ajax applications cannot be searched; Ajax elements and advanced features can be used within searchable applications.

3.3 Solution to overcome challenges

Here are some solutions and best practices to overcome those challenges:

- **Cross-Browser Compatibility**
  Use a JavaScript library or framework like jQuery or Axios to handle AJAX requests. These libraries abstract away many cross-browser compatibility issues.

- **Handling Asynchronous Nature**
  Utilize Promises or async/await syntax to handle asynchronous code more effectively.Make use of callbacks or promises for handling the results of AJAX requests to maintain code readability and manage flow.

- **SEO and Bookmarking**
  Use techniques like Progressive Enhancement or Isomorphic JavaScript to ensure that essential content is accessible to search engines and users with disabled JavaScript.Implement proper URL management for AJAX-driven content to enable bookmarking and back/forward navigation.

- **Performance Optimization**
  Minimize the amount of data transferred by sending only necessary information in AJAX requests.Implement caching mechanisms for repeated AJAX requests to reduce server load and improve response times.Consider lazy loading for content that is not immediately visible on the page.

4 Challenges and Limitations of AJAX in Web Applications

4.1 Security

- **Cross-Site Scripting (XSS):** Malicious scripts injected through AJAX requests can compromise user data and website security. Careful data validation and sanitization are crucial.
- **Cross-Site Request Forgery (CSRF):** Attackers can use AJAX to forge user requests, potentially performing unauthorized actions. Implementing anti-CSRF tokens and proper authentication is essential.

4.2 Search Engine Optimization (SEO)

- **Dynamically Loaded Content:** Search engines might not index content loaded dynamically via AJAX, impacting SEO. Server-side rendering (SSR) or pre-rendering techniques can help.
- **Deep Linking:** Bookmarking or sharing links to dynamically loaded content might not display the updated state, hindering user experience and hindering search engines from understanding the content.

4.3 Browser Compatibility

- **Older Browsers:** Not all browsers support JavaScript or XMLHttpRequest objects, limiting AJAX functionality. Providing fallbacks or graceful degradation is necessary.
- **Different JavaScript Engines:** Variations in JavaScript engine behavior across browsers can lead to inconsistencies and bugs. Thorough testing across different browsers is essential.

5 Solutions and Best Practices

5.1 Security

- Implement robust input validation and sanitization to prevent malicious script injection.
- Use HTTPS for secure communication and data encryption.
- Implement anti-CSRF tokens and proper authentication mechanisms.
- Regularly update dependencies to address known vulnerabilities.

5.2 SEO

- Utilize server-side rendering (SSR) or pre-rendering techniques to ensure search engines can index content.
- Provide static HTML content for the initial page load and enhance crawlability.
- Implement dynamic meta tags to communicate content changes to search engines.
- Use descriptive URLs for dynamic content.

5.3 Browser Compatibility

- Test thoroughly across different browsers and devices.
- Provide fallbacks or alternative solutions for older browsers using techniques like feature detection or graceful degradation.
- Leverage polyfills to fill in gaps in browser support for specific features.

AJAX implementation

```
1
2  async function fetchData(apiEndpoint) {
3      return new Promise((resolve, reject) => {
4        const xhr = new XMLHttpRequest();
5        xhr.open("GET", `https://dummyjson.com/products${apiEndpoint}`,
     true);
6        xhr.onprogress = function () {
7          console.log("loading");
8        };
9        xhr.onload = function () {
10         const arr = [];
11         if (xhr.status === 200) {
12           const arr = JSON.parse(xhr.responseText);
13           resolve(arr);
14         } else {
15           reject(`Error fetching data. Status: ${xhr.status}`);
16         }
17       };
18
19       xhr.onerror = function () {
20         reject(`Error fetching data. Status: ${xhr.status}`);
21       };
22
23       xhr.send();
24     });
25 }
```