

Web Technologies

Unit-IV

1. Explain XML structure with an example.

XML (eXtensible Markup Language) is a markup language designed to store and transport data. It is both human-readable and machine-readable.

Key Characteristics of XML:

- User-defined tags (unlike HTML which has predefined tags)
- Hierarchical data structure (tree-like)
- Supports nesting and attributes
- Strict syntax rules (well-formed and valid XML)

XML Structure:

An XML document typically includes:

1. **XML Declaration**
2. **Root Element**
3. **Child Elements**
4. **Attributes (optional)**
5. **Text content or nested elements**

Example XML Document:

```
<?xml version="1.0" ?>
```

```
<library>
```

```
  <book id="b001">
```

```
    <title>Internet & World Wide Web: How to Program</title>
```

```
    <author>H. M. Deitel</author>
```

```
    <author>P. J. Deitel</author>
```

```
    <edition>Fourth Edition</edition>
```

```
    <publisher>Pearson</publisher>
```

```
  </book>
```

```
  <book id="b002">
```

```
    <title>XML for Beginners</title>
```

```

    <author>John Smith</author>

    <edition>Second Edition</edition>

    <publisher>ABC Publications</publisher>

</book>

</library>

```

XML Part	Description
<?xml version="1.0" encoding="UTF-8"?>	Declaration – Declares version and encoding of the XML file
<library>	Root element – All other elements are nested inside
<book>	Parent/Container element – Represents one book
id="b001"	Attribute – Gives additional information about the element
<title>...</title>	Child element – Contains the title of the book
<author>...</author>	Multiple authors as child elements
Other elements like <edition>, <publisher>	Represent additional information about the book

Well-Formed Rules:

an XML document must follow these **well-formedness rules**:

- Must have exactly one root element
- Tags must be properly nested
- Every start tag must have a corresponding end tag
- Attribute values must be quoted

2. Explain XML DTD and XML Schema

Both DTD (Document Type Definition) and XML Schema are used to define the structure and rules of an XML document. They help in validating that an XML file follows the expected format. Under XML Validation, where DTD is introduced first (being older and simpler), followed by XML Schema (which is more powerful and flexible).

XML DTD (Document Type Definition)

Purpose:

A **DTD** defines the **legal structure** of an XML document — what elements can appear, in what order, and how they are nested.

✚ Types:

- **Internal DTD** – Defined within the XML file
- **External DTD** – Stored in a separate file

Example with Internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE library [
  <!ELEMENT library (book+)>
  <!ELEMENT book (title, author+, edition, publisher)>
  <!ATTLIST book id ID #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT edition (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
]>
<library>
  <book id="b001">
    <title>Internet & World Wide Web: How to Program</title>
    <author>H. M. Deitel</author>
    <author>P. J. Deitel</author>
    <edition>Fourth Edition</edition>
    <publisher>Pearson</publisher>
  </book>
</library>
```

- `<!ELEMENT library (book+)>` – A library must have one or more book elements.
- `<!ATTLIST book id ID #REQUIRED>` – Each book must have an id attribute.
- `#PCDATA` – Parsed Character Data (text).

XML Schema (XSD – XML Schema Definition)

Purpose:

XML Schema is an XML-based alternative to DTDs. It offers **stronger data typing, namespace support**, and is more extensible.

Example Schema (XSD file):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="author" type="xs:string"
maxOccurs="unbounded"/>
              <xs:element name="edition" type="xs:string"/>
              <xs:element name="publisher" type="xs:string"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

XML File Referencing the Schema:

```
<?xml version="1.0"?>
<library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="library.xsd">
  <book id="b001">
    <title>Internet & World Wide Web: How to Program</title>
    <author>H. M. Deitel</author>
    <author>P. J. Deitel</author>
    <edition>Fourth Edition</edition>
    <publisher>Pearson</publisher>
  </book>
</library>
```

Differences between XML DTD and XML Schema :

Feature	DTD	XML Schema (XSD)
Syntax	Not XML	Written in XML
Data Types	Not supported	Supported (e.g., string, int)
Namespaces	Not supported	Supported
Extensibility	Limited	High
Industry Adoption	Legacy	Widely used today

3. Explain XSL.

XSL is a family of languages used to **transform and style XML documents**. It allows XML data to be displayed in a more human-readable form, typically converting it into HTML or another XML format. **XSLT (XSL Transformations)** is the most important component of XSL.

Components of XSL:

1. **XSLT (XSL Transformations)** – Used to transform XML data into HTML, text, or another XML format.
2. **XPath** – Used within XSLT to navigate through elements and attributes in XML.
3. **XSL-FO (Formatting Objects)** – Used for formatting XML documents for output (like PDFs), but less commonly used.

XSLT Example

Let's say we have an XML file of book data, and we want to display it as an HTML table using XSLT.

XML File (books.xml):

```
<?xml version="1.0"?>
<library>
  <book>
    <title>Internet & World Wide Web: How to Program</title>
    <author>H. M. Deitel</author>
    <author>P. J. Deitel</author>
    <edition>Fourth Edition</edition>
```

```

        <publisher>Pearson</publisher>
    </book>
</library>
XSLT Stylesheet (books.xsl):
xml
CopyEdit
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
        <html>
        <body>
            <h2>Book Library</h2>
            <table border="1">
                <tr>
                    <th>Title</th>
                    <th>Authors</th>
                    <th>Edition</th>
                    <th>Publisher</th>
                </tr>
                <xsl:for-each select="library/book">
                    <tr>
                        <td><xsl:value-of select="title"/></td>
                        <td>
                            <xsl:for-each select="author">
                                <xsl:value-of select="."/>
                                <xsl:if test="position() != last()">, </xsl:if>
                            </xsl:for-each>
                        </td>
                        <td><xsl:value-of select="edition"/></td>

```

```

        <td><xsl:value-of select="publisher"/></td>

    </tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

```

Linking XML to XSLT:

In your XML file, include the reference to the stylesheet:

```
<?xml-stylesheet type="text/xsl" href="books.xsl"?>
```

Important Concepts:

Concept	Explanation
<xsl:template match="/">	Root template – the starting point for transformation
<xsl:for-each>	Loops through XML elements (like each <book>)
<xsl:value-of>	Extracts and displays data
<xsl:if>	Used for conditional logic (e.g., adding commas between authors)
Output	HTML table representation of XML content

Why Use XSLT?

- Allows **separation of content (XML)** from **presentation (XSL)**
- Helps in dynamically generating HTML views from XML data
- Supports **reusability** and **modular web design**

4. Explain Document Object Model(DOM)?

The **XML DOM** provides a standard way to **access and manipulate XML documents**. It models the entire document as a **tree of objects (nodes)** that can be dynamically modified using programming languages like **JavaScript** or **Java**.

Key Points:

1. Tree Structure Representation

- XML documents are represented as a **hierarchical tree** of nodes.
- Each node corresponds to an **element, attribute, text**, or the **document itself**.

2. Node Types

- Common node types include:
 - **Document Node**: Represents the entire document.
 - **Element Node**: Represents each XML tag.
 - **Attribute Node**: Represents attributes within elements.
 - **Text Node**: Represents the actual content between tags.

3. Root Node

- Every XML DOM tree starts with a **single root element**.
- From this node, the entire document structure can be accessed or manipulated.

4. Navigation Methods

- You can traverse the XML DOM using methods like:
 - `getElementById()`
 - `getElementsByTagName()`
 - `childNodes`, `parentNode`, `nextSibling`, etc.

5. Live Representation

- The DOM is a **live object model**, meaning changes to the DOM are reflected immediately in the document view or structure.

6. Programming Interfaces

- DOM APIs can be used with:
 - **JavaScript** (in web browsers)
 - **Java** (via JAXP or DOM parsers)
 - Other languages like Python, PHP, etc.

Simple Example:

XML File:

```
<library>
  <book id="b001">
    <title>Internet & World Wide Web</title>
    <author>H. M. Deitel</author>
  </book>
</library>
```


JavaScript Using DOM:

```
let xmlDoc = parser.parseFromString(xmlString, "text/xml");  
let titles = xmlDoc.getElementsByTagName("title");  
alert(titles[0].childNodes[0].nodeValue); // Outputs: Internet & World  
Wide Web
```

5. Explain the key differences between traditional Web Application and Ajax-based Web Application

Traditional Web Application:

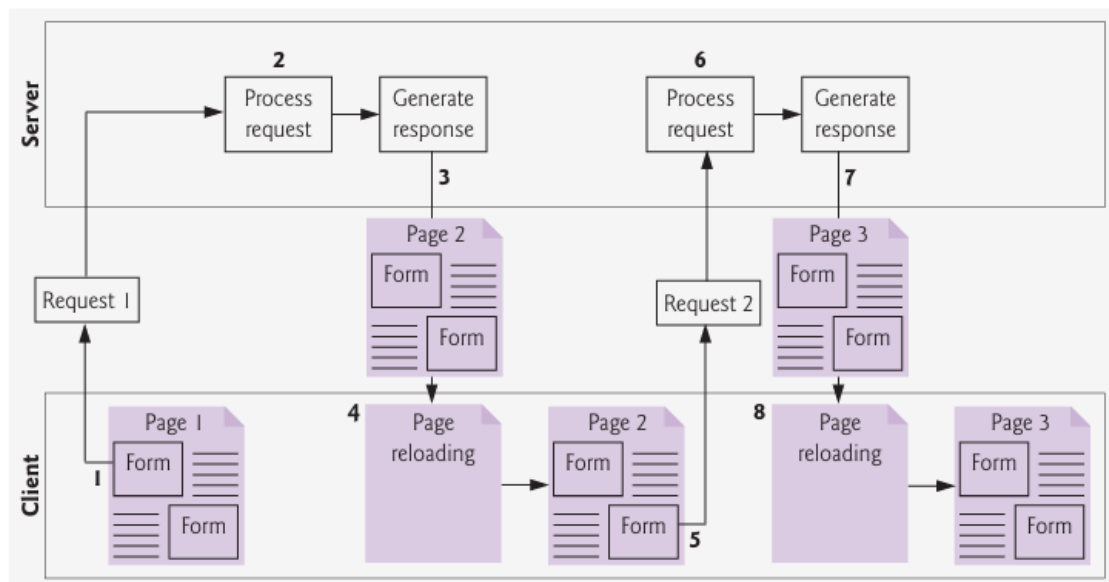
Step 1: Form Submission: The user begins by filling out a form, such as a registration form, entering details like name, email, and password. Once completed, the user submits the form, typically by clicking a "Submit" button. At this point, the browser prepares to send the form data to the server.

Step 2: Sending the Request : After submission, the browser sends an HTTP request—usually a POST request—to the web server. This request contains all the user-entered data. During this process, the request is synchronous, meaning the browser pauses interaction and waits for the server's response.

Step 3: Server Processing : The server receives the request and begins processing it. This includes validating the submitted data, checking business rules (e.g., if the email is already registered), and interacting with a database if needed. After processing, the server generates a new HTML page to return to the client.

Step 4: Loading the Response: The browser receives the full HTML response and replaces the current page with the new one. This causes the browser to go blank temporarily while loading the new content, which can feel slow and unresponsive, especially on slower connections.

Step 5-8: During this synchronous cycle, the user cannot interact with the web page while the request is being processed. This waiting time, often due to network latency or server delays, led users to refer to the web humorously as the "World Wide Wait." If another form is submitted, the same steps repeat.



Classic web application reloading the page for every user interaction.

Ajax Web Application:

Step 1: Creating the XMLHttpRequest Object: AJAX (Asynchronous JavaScript and XML) introduces a new layer between the client and server to enhance user experience. When a user interacts with an AJAX-enabled web page—for example, by entering data or clicking a button—the browser does not immediately reload the page. Instead, the client creates an XMLHttpRequest object. This object is responsible for managing the communication between the client and the server.

Step 2: Sending the Request Asynchronously: The XMLHttpRequest object sends the request to the server without interrupting the user's interaction with the web page. Unlike traditional synchronous requests, AJAX requests are **asynchronous**, meaning they do not block the user interface. This allows the user to continue interacting with the web page while the server processes the request in the background.

Steps 3 & 4: Handling Additional Requests : As the server processes the initial request, the user may continue interacting with the application. These additional interactions can result in more AJAX requests being sent to the server. This means multiple asynchronous requests can be handled concurrently—without reloading or disturbing the current state of the page.

Step 5: Server Responds and Client Callback Executes: Once the server responds to a request, the XMLHttpRequest object triggers a **callback function**—a JavaScript function defined by the developer to handle the server's response. This function processes the returned data, which could be in XML, JSON, or plain text format.

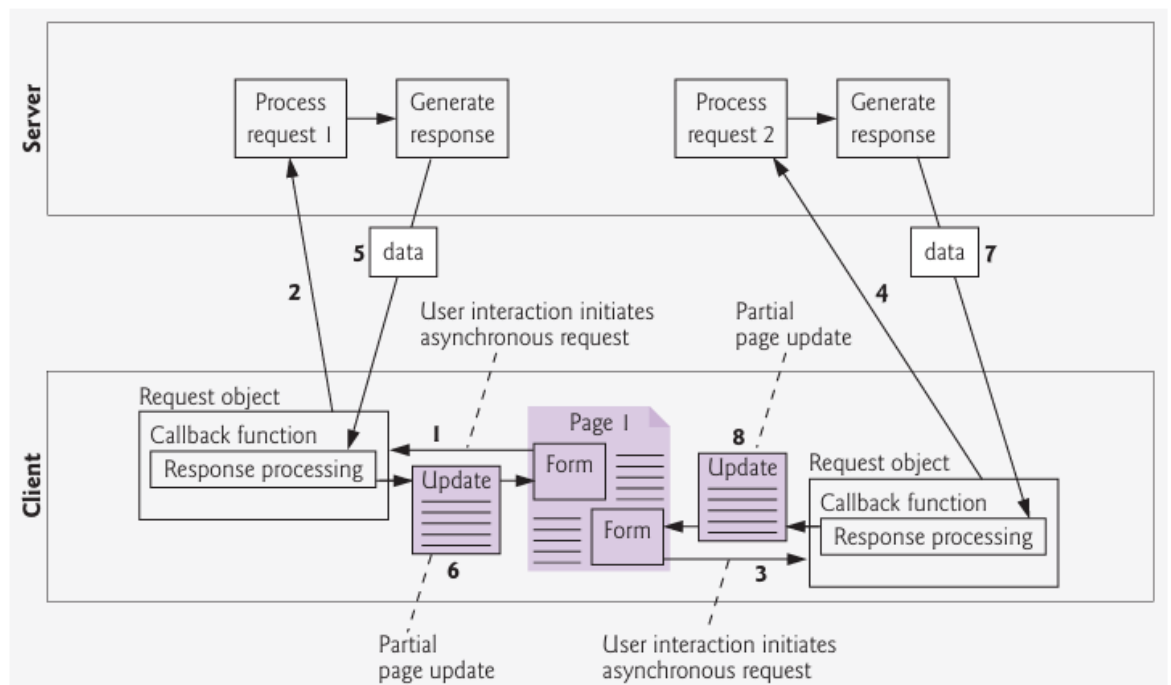
Step 6: Partial Page Update: The callback function updates a specific part of the web page—often a div or a form field—based on the server's response. This is known as a **partial page update**, and it enables dynamic, real-time changes without refreshing the entire page. For example, a user

might see search results appear below a search bar instantly, or a notification might appear without reloading.

Step 7: Concurrent Processing and Multiple Updates: While the client is updating one part of the page with a server response, the server may simultaneously be responding to a second request. At the same time, the client may also initiate another request. This parallelism allows AJAX applications to be highly interactive and **feel more like desktop applications** in terms of responsiveness.

Advantage of AJAX

The key advantage of AJAX is that it avoids full-page reloads. Instead, only small parts of the page are updated in response to user actions. This leads to a smoother, faster, and more responsive user experience, allowing modern web applications to behave much like native desktop software.



Ajax-enabled web application interacting with the server asynchronously.

6. Explain Rich Internet Applications(RIAs) with AJAX.

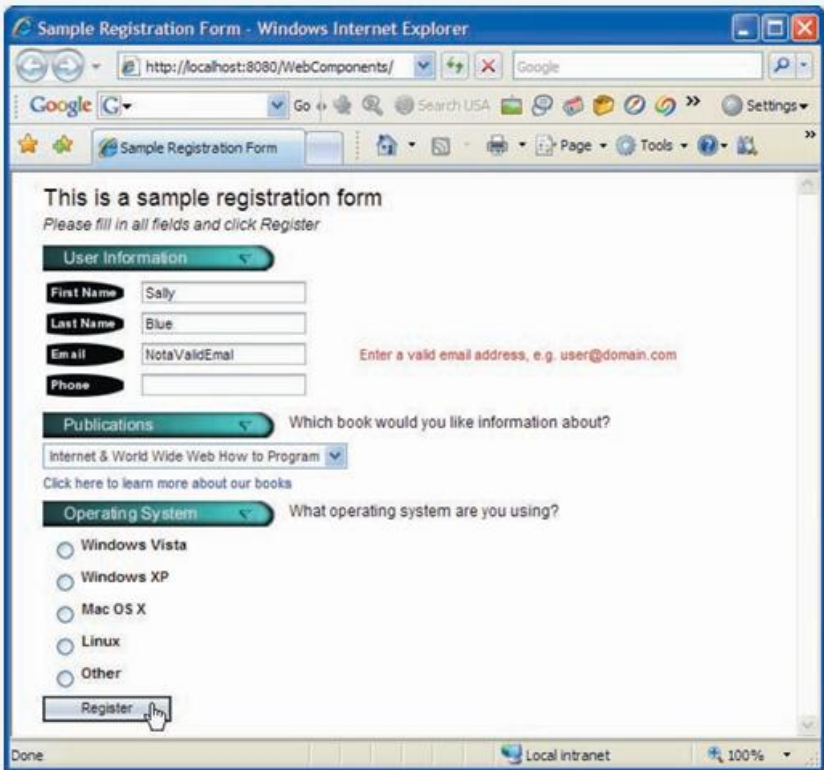
In a traditional web application, when a user fills out a form—such as a registration form containing fields like first name, last name, email address, and telephone number—they typically click a **Register** or **Submit** button to send the entire form to the server. At this point, the browser submits all data to the server for validation. During this synchronous process, the user cannot interact with the page while the server processes the form data. If the server detects any errors (e.g., an invalid email or phone number), it generates a new page showing those errors and sends it back to the client. The browser then reloads the page and displays the error messages. The user must correct the issues and resubmit the form. This cycle repeats until all the data is valid and successfully stored on the server. Unfortunately, this process can be slow and

frustrating because the entire page reloads every time invalid data is submitted.

In contrast, **AJAX-enabled forms** offer a much more interactive and responsive experience. Instead of sending the entire form for validation after submission, each input field can be validated **dynamically** and **asynchronously**—as the user is typing or moving from one field to another. For example, if a form requires a **unique email address**, an AJAX request can be triggered automatically as soon as the user enters the email and moves to the next field. This request checks the email's availability on the server in the background, without interrupting the user. If the email is already in use, the server responds with an error message that is immediately displayed next to the email field, allowing the user to correct the issue instantly.

This kind of **field-level validation** eliminates the need for multiple full-form submissions and reloads, making the form completion process much faster and less frustrating. Moreover, AJAX can also enhance the form's intelligence—for example, by using the entered **ZIP code** to automatically retrieve and fill in related fields like **city** and **state**, based on server data. This not only speeds up the form completion process but also reduces the chances of user error.

In summary, AJAX transforms static, reload-heavy forms into dynamic, user-friendly interfaces by providing **real-time validation** and **partial updates**, significantly improving usability and reducing wait times.



The screenshot shows a web browser window titled "Sample Registration Form - Windows Internet Explorer". The address bar shows "http://localhost:8080/WebComponents/". The page content includes a heading "This is a sample registration form" and a subheading "Please fill in all fields and click Register". The form is divided into three sections: "User Information", "Publications", and "Operating System". In the "User Information" section, the "First Name" field is filled with "Sally", the "Last Name" field is filled with "Blue", the "Email" field is filled with "NotaValidEmail", and the "Phone" field is empty. A red error message "Enter a valid email address, e.g. user@domain.com" is displayed next to the "Email" field. In the "Publications" section, the "Which book would you like information about?" dropdown menu is set to "Internet & World Wide Web How to Program". In the "Operating System" section, the "What operating system are you using?" question has five radio button options: "Windows Vista", "Windows XP", "Mac OS X", "Linux", and "Other". The "Register" button is at the bottom of the form. The browser's status bar at the bottom shows "Done", "Local intranet", and "100%".

Ajax-enabled form shows errors asynchronously when user moves to another field.

7. Explain XMLHttpRequest with AJAX example.

The XMLHttpRequest object (which resides on the client) is the layer between the client and the server that manages asynchronous requests in Ajax applications. This object is supported on most browsers, though they may implement it differently—a common issue in JavaScript programming. To initiate an asynchronous request, you create an instance of the XMLHttpRequest object, then use its open method to set up the request and its send method to initiate the request.

an Ajax application in which the user interacts with the page by moving the mouse over book-cover images. We use the onmouseover and onmouseout events to trigger events when the user moves the mouse over and out of an image, respectively. The onmouseover event calls function getContent with the URL of the document containing the book's description. The function makes this request asynchronously using an XMLHttpRequest object. When the XMLHttpRequest object receives the response, the book description is displayed below the book images. When the user moves the mouse out of the image, the onmouseout event calls function clearContent to clear the display box. These tasks are accomplished without reloading the page on the client.

```
// Create a new XMLHttpRequest object
var xhr = new XMLHttpRequest();

// Configure it: GET-request for the URL /ajax_example.txt
xhr.open('GET', 'ajax_example.txt', true);

// Set up a function that will be called when the request completes
xhr.onreadystatechange = function () {
    // Check if the request is complete (readyState 4)
    if (xhr.readyState == 4) {
        // Check if the request was successful (status 200)
        if (xhr.status == 200) {
            // Output the response text to the console
            console.log(xhr.responseText);
        } else {
            // Handle error, if any
            console.error("Request failed with status: " + xhr.status);
        }
    }
}
```

```

    }
}
};

// Send the request to the server
xhr.send();

```

8. Explain the process of creating a Full scale AJAX enabled application.

Creating a full-scale AJAX-enabled application involves a few steps where you use JavaScript, XMLHttpRequest (or Fetch API), HTML, and a server-side language to build dynamic web applications that can fetch and send data asynchronously.

Here's a high-level breakdown of the steps to create an AJAX-enabled application:

Step-by-Step Process for Creating an AJAX Application:

1. Define the Requirements of the Application

First, outline the functionality of your application. An AJAX-enabled web application allows portions of the page to update dynamically without needing to reload the entire page. For example, an online chat system, dynamic form validation, or live search functionality could be built using AJAX. Define the server-side functionality that your application will require (e.g., fetching data from a database, submitting forms, etc.).

2. Create the Basic HTML Structure

This includes the basic layout of your web page and the user interface (UI). For example, you may have a text box, a submit button, and a display area for showing results.

HTML file

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>AJAX Example</title>
  <script src="ajax.js"></script> <!-- Link to the external JS file for AJAX
-->
</head>
<body>

```

```

<h2>AJAX Example</h2>
<input type="text" id="query" placeholder="Enter a search term">
<button onclick="sendRequest()">Search</button>
<div id="result"></div>
</body>
</html>

```

3. Create JavaScript for Making AJAX Requests

Use JavaScript to create the XMLHttpRequest object or the newer Fetch API to make asynchronous HTTP requests to the server. Here's an example of how to use XMLHttpRequest:

javascript

```

// File: ajax.js
function sendRequest() {
    var query = document.getElementById('query').value; // Get the user
    input
    var xhr = new XMLHttpRequest(); // Create a new XMLHttpRequest
    object

    // Setup the GET request (using query parameter)
    xhr.open('GET', 'search.php?query=' + query, true);

    // Define a function to handle the response
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
            document.getElementById('result').innerHTML =
xhr.responseText;
        }
    };

    // Send the request to the server
    xhr.send();
}

```

4. Write Server-Side Code

On the server, you need to handle the request made by the client and send the appropriate response back. You can use any server-side language such as PHP, Node.js, Python, etc.

5. Testing the Application

Once the client-side (HTML, JavaScript) and server-side (e.g., PHP) code is ready, test the application.

Enter a search term in the input box, click the "Search" button, and check if the result is displayed dynamically below the input box without a full page reload.

6. Enhance the Application

As the application grows, you may want to add more features like form validation, real-time updates, etc.

You might use advanced JavaScript techniques like handling different types of HTTP requests (GET, POST), managing errors, and optimizing for performance.

7. Debugging and Optimizing the Application

During the development of an AJAX application, you need to debug the client-side JavaScript and the server-side code.

Utilize browser developer tools to inspect network requests, track down errors, and analyze the flow of data between the client and the server.

8. Final Testing and Deployment

After all functionality is in place, perform final testing to ensure everything works smoothly across different browsers and devices.

Deploy your application on a web server for public access.