Ajax

5.7.1 What is Ajax?

Ajax, which stands for "Asynchronous JavaScript and XML," is a web development technique used to create more interactive and responsive web applications. It allows web pages to retrieve and exchange data with a web server in the background, without requiring a full page reload. This asynchronous communication enables parts of a web page to be updated or refreshed independently, resulting in a smoother and more dynamic user experience.

It is a set of web development techniques that uses various web technologies on the client-side to create asynchronous web applications. With AJAX, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page.

AJAX is used in a wide variety of web applications, including:

- Google Suggest
- Gmail
- Google Maps
- Facebook
- Twitter
- Amazon
- Wikipedia

5.7.2 Use of Ajax

i. **Dynamic Content Loading:** Ajax enables the dynamic loading of content without refreshing the entire web page. For example, it can be used to load new posts or comments in a social media feed when the user scrolls down, providing a seamless browsing experience.

ii. **Form Validation:** Ajax can validate form data in real-time as users fill out forms. It can check for errors, such as invalid email addresses, and provide instant feedback to users, improving the user experience.

iii. **Autosuggest and Autocomplete:** Ajax can be used to create autosuggest and autocomplete features in search bars and input fields. As users type, Ajax sends requests to the server to fetch relevant suggestions, which are then displayed in real-time.

iv. **Real-Time Chat and Messaging:** Many messaging apps and chat systems use Ajax to send and receive messages without requiring manual refreshes. This provides real-time communication between users.

v. **Live Updates and Notifications:** Websites often use Ajax to provide live updates and notifications. For example, a stock trading platform can use Ajax to show real-time price changes, or a news website can display breaking news updates without reloading the page.

vi. **Infinite Scrolling:** Ajax is frequently used to implement infinite scrolling, where new content is loaded as users scroll down a page. This is commonly seen in social media feeds, image galleries, and product catalogs.

vii. **Data Visualization:** Ajax can fetch data from a server for creating interactive charts, graphs, and data visualizations. It's often used in conjunction with JavaScript charting libraries.

viii. **Content Filtering and Sorting:** E-commerce websites use Ajax to let users filter and sort products without reloading the entire product list. This makes it easier for users to find the products they want.

ix. **Geo Location Services:** Websites and apps can use Ajax to fetch location-based data, such as maps, weather updates, and nearby points of interest.

x. **Database Interaction:** Ajax can interact with databases on the server, enabling users to query and update data without refreshing the entire page. This is often seen in content management systems (CMS) and web-based applications.

xi. **File Upload and Download:** Ajax can handle file uploads and downloads, allowing users to upload files to a server or download files without leaving the current page.

xii. **Gaming:** Ajax can be used to create simple web-based games, handling game logic, user interactions, and real-time updates.

5.7.3 How to send data with Ajax Request?

i. Create an XMLHttpRequest object.
ii. Open a connection to the server by calling the open() method.
iii. Set the request method to POST.
iv. Set the request headers, if needed.
v. Set the request body to the data that you want to send.
vi. Send the request by calling the send() method.

```
// Step 1: Create an XMLHttpRequest object

var xhr = new XMLHttpRequest();


// Step 2: Open a connection to the server

xhr.open('POST', 'https://jsonplaceholder.typicode.com/posts', true);


// Step 5: Set the request body to the data you want to send

var data = {

  title: 'Sample Post',

  body: 'This is the content of the sample post.',

  userId: 1

};


// Convert the data to a JSON string

var jsonData = JSON.stringify(data);


// Step 6: Send the request

xhr.send(jsonData);


// Step 7: Listen for the response by attaching an event listener to the onload
property

xhr.onload = function () {
  // Step 8: Once the response is received, parse the response data and use it
  if (xhr.status === 201) { // 201 indicates a successful resource creation (HTTP
status code)
    var response = JSON.parse(xhr.responseText);
    console.log('Post ID:', response.id);
  } else {
    console.error('Error:', xhr.status, xhr.statusText);
  }
};

// Handle any network errors
xhr.onerror = function () {
  console.error('Network error');
};
```

5.7.4 Difference between get, post, put, delete method

i. GET Method:

Purpose: Used for requesting data from the server.

Data in Request: Data can be sent as URL parameters in the request's query string.

Safety: Considered safe because it should not have the side effect of modifying data on the server.

Idempotent: Multiple identical GET requests should produce the same result.

Caching: Responses can be cached by browsers and intermediaries.

Example Use Cases: Fetching a webpage, retrieving a list of items, searching, and viewing data.

ii. POST Method:

Purpose: Used for sending data to the server to create a new resource.

Data in Request: Data is sent in the request body. It can be in various formats, including form data, JSON, XML, etc.

Safety: Not considered safe, as it may result in the creation of a new resource or modify existing resources.

Idempotent: It's not idempotent; multiple identical POST requests may create multiple resources.

Caching: Responses are typically not cached by browsers and intermediaries.

Example Use Cases: Submitting a form, creating a new user account, or adding a new record to a database.

iii. PUT Method:

Purpose: Used for updating an existing resource on the server.

Data in Request: Data is sent in the request body and is used to update an existing resource at the specified URL.

Safety: Not considered safe, as it may modify data on the server.

Idempotent: It's idempotent, meaning that multiple identical PUT requests should have the same effect as a single request.

Caching: Responses are typically not cached by browsers and intermediaries.

Example Use Cases: Updating a user's profile, modifying an existing record in a database.

iv. DELETE Method:

Purpose: Used for requesting the removal of a resource from the server.

Data in Request: The resource to delete is typically identified in the URL, and no request body is required.

Safety: Not considered safe, as it results in the removal of a resource.

Idempotent: It's idempotent; multiple identical DELETE requests should have the same effect as a single request.

Caching: Responses are typically not cached by browsers and intermediaries.

Example Use Cases: Deleting a user account, removing a record from a database.

5.7.5 JSON data

JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

JSON is a text-based data format following JavaScript object syntax, which was popularized by Douglas Crockford. Even though it closely resembles JavaScript object literal syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.

JSON exists as a string — useful when you want to transmit data across a network. It needs to be converted to a native JavaScript object when you want to access the data. This is not a big issue — JavaScript provides a global JSON object that has methods available for converting between the two.

JSON Structure: -

As described above, JSON is a string whose format very much resembles JavaScript object literal format. You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals. This allows you to construct a data hierarchy.

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john@example.com",
  "isSubscribed": true,
  "address": {
    "street": "123 Main St",
    "city": "Cityville",
    "zipCode": "12345"
  },
  "hobbies": ["Reading", "Hiking", "Cooking"]
}
```

5.7.6 Serialization & De-serialization

Serialization and deserialization are processes used in computer science and data exchange to convert data between different formats or representations, typically for the purpose of storage, transmission, or interchange between systems. These processes are essential in various contexts, including data storage, network communication, and data transfer between different programming languages.

Serialization is the process of converting an object into a stream of bytes so that it can be stored or transmitted. Deserialization is the reverse process of converting a stream of bytes back into an object.

serialization and deserialization are used to send and receive data from a server. For example, if you are using an Ajax request to submit a form, the form data will need to be serialized before it can be sent to the server. The server will then deserialize the form data before it can be processed.

- **Serialization: -**

Serialization is the process of converting data structures or objects into a format that can be easily stored or transmitted and later reconstructed. The primary goal of serialization is to make data portable and independent of its original format. Serialized data is often represented as a stream of bytes, text, or other suitable formats.

```
var object = {
  name: "John Doe",
  age: 30
};


var json = JSON.stringify(object);
```

- **De-serialization: -**

De-serialization is the reverse process of serialization. It involves taking serialized data and reconstructing it into its original data structure or object. The goal of de-serialization is to recreate the original data or object with the same structure and content as it had before serialization.

```
var json = '{ "name": "John Doe", "age": 30 }';


var object = JSON.parse(json);
```

❖ jQuery Ajax Methods: -

Method	**Description**
$.ajax()	Performs an async AJAX request
$.ajaxPrefilter()	Handle custom Ajax options or modify existing options before each request is sent and before they are processed by $.ajax()
$.ajaxSetup()	Sets the default values for future AJAX requests
$.ajaxTransport()	Creates an object that handles the actual transmission of Ajax data
$.get()	Loads data from a server using an AJAX HTTP GET request
$.getJSON()	Loads JSON-encoded data from a server using a HTTP GET request
$.getScript()	Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request
$.param()	Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)
$.post()	Loads data from a server using an AJAX HTTP POST request
ajaxComplete()	Specifies a function to run when the AJAX request completes

ajaxError()	Specifies a function to run when the AJAX request completes with an error
ajaxSend()	Specifies a function to run before the AJAX request is sent
ajaxStart()	Specifies a function to run when the first AJAX request begins
ajaxStop()	Specifies a function to run when all AJAX requests have completed
ajaxSuccess()	Specifies a function to run when an AJAX request completes successfully
load()	Loads data from a server and puts the returned data into the selected element
serialize()	Encodes a set of form elements as a string for submission
serializeArray()	Encodes a set of form elements as an array of names and values