Module 4

Q. 1 What is RIA?. Give characteristics of RIA.

1. RIA Stands for Rich Internet Application.
2. Rich Internet Application (RIA) is a application designed in such a way that it delivers the Functionalities and features same as desktop appications delivered over the web.
3. RIA does not require software installation as it normally runs inside a web browser.

Characteristics of RIA:

1. Interactivity: RIAs offer a high level of interactivity, allowing users to manipulate data and interfaces in real time. This includes drag-and-drop functionality, dynamic content updates, and responsive user interfaces.
2. Rich User Interface (UI): They utilize rich graphics, animations, and multimedia elements, making the application visually appealing and user-friendly. This often involves technologies like HTML5, CSS3, and JavaScript frameworks.
3. Asynchronous Communication: RIAs frequently use AJAX (Asynchronous JavaScript and XML) to exchange data with the server without reloading the entire web page. This allows for smoother interactions and quicker updates.
4. Client-Side Processing: A significant amount of processing occurs on the client side, reducing the load on the server and providing a faster user experience. This is often accomplished through JavaScript and related libraries.
5. Offline Capabilities: Some RIAs can function partially or fully offline, storing data locally and syncing with the server when a connection is available. This is achieved through technologies like Service Workers and local storage.
6. Consistency of look and feel in Rich Internet Applications (RIAs) means making sure that the design and layout are the same throughout the app. This helps users feel comfortable and understand how to use it easily.
7. Cross-Platform Compatibility: RIAs are typically designed to work across various platforms and devices (desktops, tablets, smartphones), providing a consistent user experience.

8. Enhanced Performance: By minimizing server calls and processing data locally, RIAs Generally, exhibit better performance compared to traditional web applications.
9. State Management: RIAs often maintain user state across sessions and interactions, allowing for a seamless user experience where data and settings persist even as users navigate through different parts of the application.

Examples of RIAs:

1. Web-based email clients (like Gmail)
2. Online document editors (like Google Docs)
3. Interactive dashboards for data visualization
4. Google Maps : adjacent maps are "Pre-fetched".

Q. 2 What is AJAX? Explain AJAX web application with neat diagram.

1. AJAX stands for asynchronous Javascript and XML
2. AJAX is not a programming language instead, it is a method of accessing data from the server asynchronously and updating the web pages without refreshing/reloading them.
3. The AJAX model allows web developers to create web applications that are able to dynamically interact with the user.
4. It will also be able to quickly make a background call to web servers to retrieve the required application data. Then update the small portion of the web page without refreshing the whole web page.
5. AJAX applications are much more faster and responsive as compared to traditional web applications.
6. It creates a great balance between the client and the server by allowing them to communicate in the background while the user is working in the foreground.

Example : Have you ever thought "How does the score of the live cricket match updates automatically in web page?" Do you need to refresh the whole page while updating score? The Answer is AJAX technology is used which communicates to and from a server/database without the need for a post back or a complete page refresh.

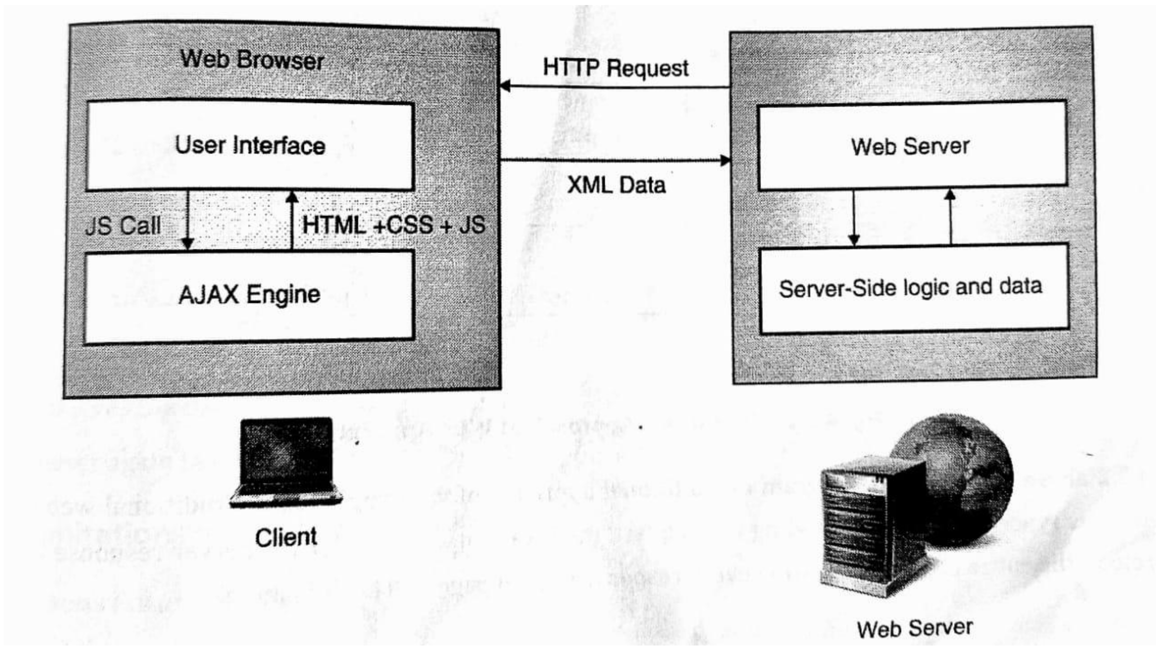

Fig : AJAX Basic Design

Steps in AJAX

1. An event occurs in a web page (the page is loaded, a button is clicked).

2. An XMLHttpRequest object is created by JavaScript.

3. The XMLHttpRequest object sends a request to a web server.

4. The server processes the request.

5. The server sends a response back to the web page.

6. The response is read by JavaScript.

7. Proper action (like page update) is performed by JavaScript.

Q. 3 What is the use of XMLHttpRequest object? Explain methods that are used to send request to server using AJAX.

1. XMLHTTPRequest object is an API which is used for fetching data from the server.
2. XMLHTTPRequest is basically used in Ajax programming.
3. It retrieve any type of data such as json, xml, text etc. It request for data in background and update the page without reloading page on client side.
4. An object of XMLHTTPRequest is used for asynchronous communication between client and server.
5. The $.ajax() method is used for the creation of XMLHTTPRequest object.

The $.ajax() does following steps in background:

1. Send data from background.
2. Receives the data from the server.
3. Update webpage without reloading the page.

Methods that are used to send request to server using AJAX:

1. open(method, url, async)	:	Specifies the type of request method: the type of request: GET or POST url: the server (file) location async: true (asynchronous) or false (synchronous)
2. send()	:	Sends the request to the server (used for GET)
3. send(string)	:	Sends the request to the server (used for POST)
4. new XMLHttpRequest()	:	It is used to create an XMLHttpRequest() object.
5. abort()	:	It is used to cancel the current request.
6. getAllResponseHeaders()	:	It is used to get the header information
7. getResponseHeader()	:	It is used to get the specific header information
8. setRequestHeader()	:	It is used to add key/value pair to the header.

Example:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Q. 4 Write a AJAX code to read from the text file and display it after clicking on a button .

```
<html>
<head>
<script type="text/javascript">
function loadXMLDoc() {
```

```
req=new XMLHttpRequest();
req.onreadystatechange=function() {
if (req.readyState==4 && req.status==200) {
document.getElementById("div1").innerHTML=req.responseText;
}
} // end of onreadstatechange function
req.open("GET","myinfo.txt",true);
req.send();
} // end of loadXMLDoc()
</script>
</head>
<body>
<div id="div1">my first ajax program</div>
<input type="button" id="button1" value="click me!" onclick="loadXMLDoc()"/>
</body>
</html>
```

Q. Draw a diagram of Ajax application model and Traditional application web model and compare them.

AJAX Application :

1. The **Client (Browser)** interacts with the **JavaScript (AJAX)** on the page.
2. **JavaScript** sends an **Asynchronous HTTP Request** to the **Web Server**.
3. The **Web Server** processes the request, interacting with the **Application Logic/Server-Side** and potentially querying the **Database**.
4. The response (partial data, such as JSON or XML) is sent back from the server to the **Client** without refreshing the entire page.
5. **JavaScript (AJAX)** processes the response and dynamically updates parts of the page, without a full reload.

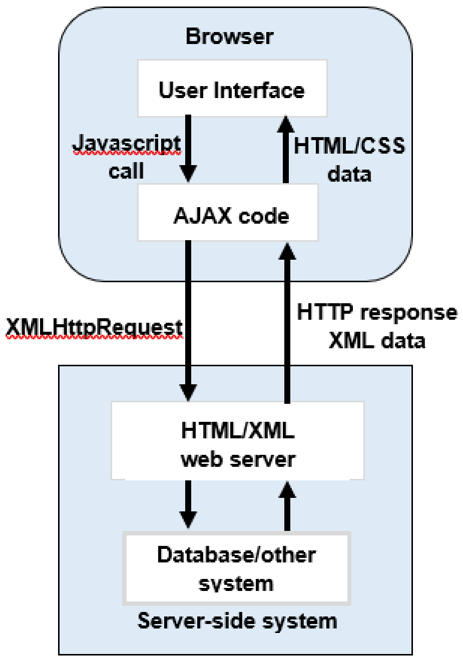

Fig : AJAX Basic Design

Characteristics:

- **Partial Updates**: Only the specific part of the page that needs updating is refreshed, rather than reloading the entire page.
- **Asynchronous**: The client can continue interacting with the page while data is being fetched in the background.
- **Faster Response Time**: Only relevant data is transferred, so the application feels more responsive.
- **Improved User Experience**: Smooth interaction without full-page reloads, making it feel more like a desktop application.

Traditional Web Application :

1. **Client (Browser)** sends an HTTP request to the **Web Server**.
2. The **Web Server** processes the request, passing it to the **Application Logic/Server-Side**.
3. The **Application Logic** may interact with a **Database** to retrieve or update data.
4. The response is generated and sent back to the **Client** via **Web Server**.
5. The client receives the full new page and renders it.

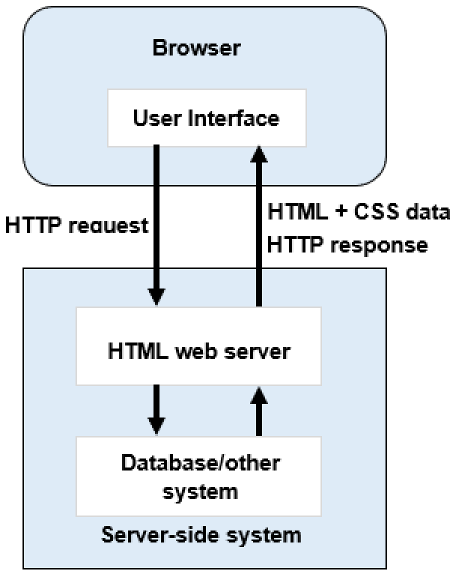

Fig : Traditional Web Application Design

Characteristics:

- **Full Page Reloads**: Every user interaction with the application requires a new page request to the server and a full page reload.
- **Slow Response Time**: Since the entire page reloads, the process can be slow and inefficient.
- **Limited User Experience**: A user must wait for the page to reload entirely, making the application feel less dynamic.

Comparison of Ajax vs. Traditional Web Application Model:

Feature	**Traditional Web Application Model**	**Ajax Application Model**
Page Reloads	Entire page reloads after each request	No page reloads; only parts of the page are updated
Response Time	Slow, as the full page needs to reload	Fast, as only necessary data is sent and received
User Experience	Static, feels like you're starting over every time	Dynamic, feels like a desktop app, smooth updates
Server Requests	Each user interaction requires a full HTTP request to the server	Multiple asynchronous requests made to the server in the background

Complexity	Simpler, no need for complex JavaScript	More complex, requires JavaScript and server-side support for asynchronous communication
Browser Interaction	User must wait for page reloads, disrupting flow	User can interact with the page while waiting for data
Bandwidth Usage	More bandwidth required as entire page is sent each time	Less bandwidth usage since only the required data is sent

Q. Explain JQuery Framework with AJAX.

1. **jQuery** is a lightweight, fast, and feature-rich JavaScript library that simplifies things like HTML document traversal and manipulation, event handling, animation, and AJAX interactions for rapid web development.
2. **AJAX** (Asynchronous JavaScript and XML) is a technique used to send and receive data asynchronously without reloading the entire web page.
3. Using **jQuery** with **AJAX** allows you to send requests to the server, retrieve data, and update parts of the web page dynamically without requiring a full page reload.
4. jQuery makes working with AJAX easier by providing convenient methods that abstract away the complexities of traditional JavaScript.

How jQuery AJAX Works

Here's how you can integrate **jQuery** with **AJAX**:

1. AJAX Request Flow

- **User Event** (click, submit, etc.) triggers the AJAX request.
- The **AJAX Request** is sent asynchronously to the server, requesting data or sending data.
- The server processes the request, interacts with the database if necessary, and returns a response (usually JSON, XML, or HTML).
- jQuery receives the response and uses it to update the webpage dynamically without refreshing the entire page.

2. Using jQuery for AJAX Requests

jQuery offers several methods for making AJAX requests, but the most commonly used ones are:

- **$.ajax()** – The most flexible method, allowing you to configure all aspects of the request.
- **$.get()** – A shorthand for making GET requests.

- **$.post()** – A shorthand for making POST requests.
- **$.getJSON()** – A shorthand for making GET requests that return JSON data.

Advantages of Using jQuery with AJAX:

1. **Ease of Use**: jQuery simplifies syntax for making AJAX calls and handling responses.
2. **Cross-Browser Compatibility**: jQuery takes care of browser differences, ensuring that your AJAX code works consistently across all major browsers.
3. **Dynamic Page Updates**: It allows you to update parts of a webpage dynamically without reloading, making the user experience smoother.
4. **Simplified Error Handling**: jQuery provides simple methods for handling errors and success scenarios.
5. **Shorthands**: Methods like $.get(), $.post(), and $.getJSON() make working with AJAX very convenient.

Commonly used JQuery Event Methods :

Event Method	**Description**	**Ajax Application Model**

click()	Triggered when the element is clicked by the user.	`$("#myButton").click(function() {`
		`  alert("Button clicked!");`
		`});`

dblclick()	Triggered when the element is double-clicked.	`$("#myDiv").dblclick(function() {`
		`  alert("Div double-clicked!");`
		`});`

focus()	Triggered when an element gains focus.	`$("input").focus(function() {`
		`  $(this).css("background-color", "lightblue");`
		`});`

blur()	Triggered when an element loses focus.	`$("input").blur(function() {`
		`  $(this).css("background-color", "");`
		`});`

mouseenter()	Triggered when the mouse enters the element.	`$("#myDiv").mouseenter(function() {`
		`  $(this).css("background-color", "lightgreen");`
		`});`

mouseleave()	Triggered when the mouse leaves the element.	`$("#myDiv").mouseleave(function() {`
		`  $(this).css("background-color", "");`
		`});`

keydown()	Triggered when a key is pressed down.	$(document).keydown(function(event) {
		    console.log("Key pressed: " + event.key);
		});

keyup()	Triggered when a key is released.	$(document).keyup(function(event) {
		    console.log("Key released: " + event.key);
		});

keypress()	Triggered when a key is pressed (for printable characters).	$("#inputField").keypress(function(event) {
		    console.log("Key pressed: " +
		String.fromCharCode(event.which));
		});

submit()	Triggered when a form is submitted.	$("form").submit(function(event) {
		    alert("Form submitted!");
		    event.preventDefault();
		});

change()	Triggered when the value of an input element changes.	$("input[type='checkbox']").change(function() {
		    if($(this).is(":checked")) {
		        alert("Checkbox is checked");
		    } else {
		        alert("Checkbox is unchecked");
		    }
		});

on()	A flexible method to handle multiple events.	$("div").on("click", function() {
		    alert("Div clicked!");
		});

mousedown()	It is triggered when the mouse button is pressed on an element.	$("#myDiv").mousedown(function() { $(this).css("border", "2px solid red"); });
mouseup()	It is triggered when the mouse button is released after being pressed on an element.	$("#myDiv").mouseup(function() { $(this).css("border", ""); });