Name : Aditya Sampath Kumar

Class : D15A

Roll no. : 01

Experiment – 9: AJAX

1) **Aim:** To study AJAX
2) **Theory:**

A. How do Synchronous and Asynchronous Requests differ?

Synchronous and asynchronous requests are two approaches used by web applications to communicate with servers, particularly when fetching or sending data. Their primary difference lies in how they handle **timing and execution flow**.

1. Synchronous Requests:

- In a **synchronous request**, the code execution **pauses** until the server responds.
- The browser **waits** for the request to complete before continuing with the next line of code.
- This blocking behavior means the user interface (UI) may become **unresponsive** during the request.
- It's generally discouraged in modern web development, especially in browsers, because it leads to a **poor user experience**.

Example Scenario:
If a webpage sends a synchronous request to fetch user data, the page may freeze while waiting for the server, and users won't be able to interact with anything until it finishes.

2. Asynchronous Requests:

- An **asynchronous request** does **not block** the code execution.
- The browser sends the request and continues executing the rest of the code **without waiting** for the response.
- A **callback function** or event listener is used to handle the server's response when it arrives.
- This improves **performance** and ensures a **smooth user experience** because the UI remains active and responsive.

Example Scenario:
When a form is submitted using AJAX asynchronously, the data is sent to the server in the background, and the user can continue interacting with the page while waiting for confirmation.

Key Differences Summarized in Words:

- Synchronous = Blocking, halts execution, bad for UX
- Asynchronous = Non-blocking, continues execution, preferred in modern web apps
- Asynchronous uses event-driven programming to respond to server replies
- Asynchronous requests are handled using XMLHttpRequest, fetch(), or async/await in JavaScript

B. Describe various properties and methods used in XMLHttpRequest Object.

Properties of XMLHttpRequest:

1. readyState
   Indicates the current state of the request (ranging from 0 to 4, where 4 means the request is complete).
2. status
   Shows the HTTP response status code (like 200 for success or 404 for not found).
3. statusText
   Provides a text description of the HTTP status (like "OK" or "Not Found").
4. responseText
   Holds the response data returned by the server as plain text.
5. responseXML
   Contains the response data as an XML document, if applicable.
6. onreadystatechange
   A property used to define a function that runs automatically whenever the request's state changes.

Methods of XMLHttpRequest:

1. open()
   Prepares the request by specifying the HTTP method, URL, and whether the request should be asynchronous.
2. send()
   Sends the request to the server. Can include data when using methods like POST.
3. setRequestHeader()
   Allows setting custom HTTP headers (like content type) before sending the request.
4. abort()
   Cancels the request if it’s still in progress.

3) Problem Statement:

Create a registration page having fields like Name, College, Username and Password (read password twice).
Validate the form by checking for
1. Usernameis not same as existing entries
2. Name field is not empty
3. Retyped password is matching with the earlier one. Prompt a
message is And also auto suggest college names.
Show the message "Successfully Registered" on the same page below the submit button, on Successfully registration. Let all the updations on the page be Asynchronously loaded. Implement the same using XMLHttpRequest Object.

4) Output:

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Registration Form</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 40px;
    }
    input, button {
      display: block;
      margin: 10px 0;
      padding: 8px;
      width: 300px;
    }
    .suggestions {
      border: 1px solid
      #ccc; max-height:
      100px; overflow-y:
      auto; width: 300px;
      background: #f9f9f9;
      position: absolute;
      z-index: 10;
    }
    .suggestion-item {
      padding: 5px;
      cursor: pointer;
    }
    .suggestion-item:hover {
      background: #ddd;
    }
    #message {
      font-weight: bold;
      margin-top:
      10px;
    }
  </style>
</head>
<body>

<h2>Registration Form</h2>
<form id="registrationForm" onsubmit="return false;">
  <input type="text" id="name" placeholder="Name" required />

  <div style="position: relative;">
        <input type="text" id="college" placeholder="College" autocomplete="off"
onkeyup="autoSuggestCollege()" />
```

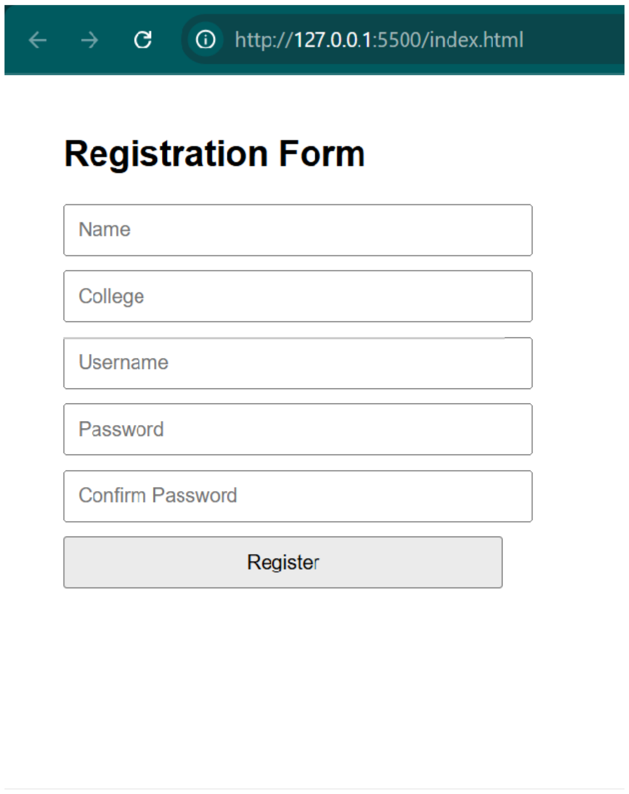
```
    <div id="suggestions" class="suggestions"></div>
  </div>

  <input type="text" id="username" placeholder="Username" required />
  <input type="password" id="password" placeholder="Password" required />
   <input type="password" id="confirmPassword" placeholder="Confirm Password"
required />

  <button onclick="submitForm()">Register</button>
</form>

<p id="message"></p>

<script>
  const existingUsernames = ['user123', 'john_doe', 'admin'];
  const colleges = [
    "Indian Institute of Technology",
    "National Institute of Technology",
    "Anna University",
    "Delhi University",
    "VIT University",
    "SRM Institute",
    "MIT College",
    "BITS Pilani"
```

```
];

function autoSuggestCollege() {
  const input = document.getElementById("college").value.toLowerCase();
  const suggestionsBox = document.getElementById("suggestions");
  suggestionsBox.innerHTML = '';

  if (input === '') return;

  const matched = colleges.filter(college =>
    college.toLowerCase().includes(input)
  );

  matched.forEach(college => {
    const div = document.createElement("div");
    div.className = "suggestion-item";
    div.innerText = college;
    div.onclick = () => {
      document.getElementById("college").value = college;
      suggestionsBox.innerHTML = '';
    };
    suggestionsBox.appendChild(div);
  });
```

```
}


// Live check for existing username

document.getElementById("username").addEventListener("blur", function () {

const username = this.value.trim();

  const message = document.getElementById("message");


  if (existingUsernames.includes(username)) {

    message.innerText = "Username already exists.";

    message.style.color = "red";

  } else {

    message.innerText = "";

  }

});


function submitForm() {

  const name = document.getElementById("name").value.trim();

  const college = document.getElementById("college").value.trim();

  const username = document.getElementById("username").value.trim();

  const password = document.getElementById("password").value;

  const confirmPassword = document.getElementById("confirmPassword").value;

  const message = document.getElementById("message");
```

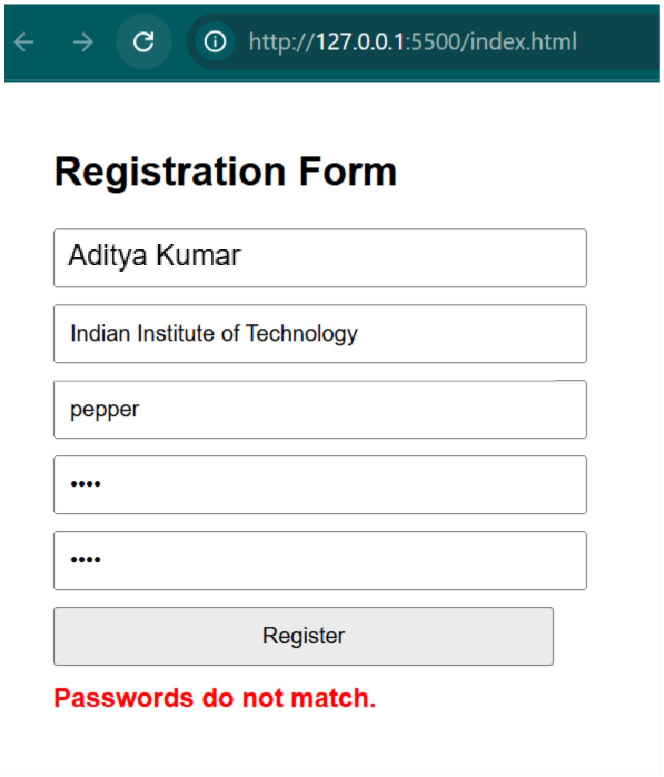
```
    // Simulate async behavior
    setTimeout(() => {
      let response = "";

      if (name === "") {
        response = "Name cannot be empty.";
      } else if (existingUsernames.includes(username)) {
        response = "Username already exists.";
      } else if (password !== confirmPassword) {
        response = "Passwords do not match.";
      } else {
        response = "Successfully Registered";
      }

      message.innerText = response;
      message.style.color = response === "Successfully Registered" ? "green" : "red";
    }, 500); // fake delay to mimic XMLHttpRequest async loading
  }
</script>

</body>
</html>
```

OUTPUT:



http://127.0.0.1:5500/index.html

Registration Form


http://127.0.0.1:5500/index.html

Registration Form

Aditya Kumar

Indian Institute of Technology

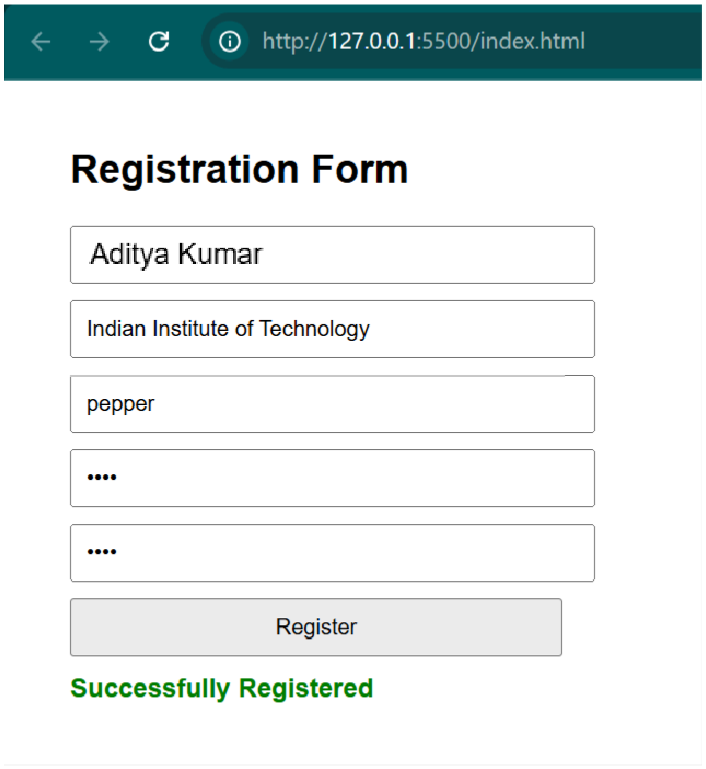
pepper

••••

••••

Register

Passwords do not match.


http://127.0.0.1:5500/index.html

Registration Form

Aditya Kumar

Indian Institute of Technology

pepper

••••

••••

Register

Successfully Registered