# Experiment – 9: AJAX

| | |
|---|---|
| **Name of Student** | Sannidhi Kailaje |
| **Class Roll No** | D15A / 22 |
| **D.O.P.** | |
| **D.O.S.** | |
| **Sign and Grade** | |

**Aim:** To study AJAX

**Theory:**
How do Synchronous and Asynchronous Requests differ?

Synchronous and Asynchronous requests refer to how tasks are executed in relation to the main program flow. In a **synchronous request**, the execution of code is paused until the request completes and a response is received from the server. This can cause the browser or application to become unresponsive, especially if the server takes a long time to respond. It is a blocking operation and is generally not recommended for modern web applications due to poor user experience.

In contrast, an **asynchronous request** allows the program to continue executing other operations while the request is still being processed in the background. Once the response is received, a callback function is triggered to handle the result. This non-blocking behavior makes asynchronous requests ideal for enhancing interactivity and responsiveness in web applications, such as updating parts of a webpage without reloading it entirely.

Describe various properties and methods used in XMLHttpRequest Object

The **XMLHttpRequest object** is a built-in JavaScript object used to interact with servers. It allows web pages to make HTTP requests to retrieve or send data without refreshing the page, enabling AJAX-based dynamic content loading.

**Common Properties:**

- **readyState**: Holds the status of the request. Values range from 0 (uninitialized) to 4 (request finished and response is ready).
- **status**: Returns the HTTP status code (e.g., 200 for success, 404 for not found).
- **statusText**: Provides a short message corresponding to the status code.
- **responseText**: Returns the response data as a string.
- **responseXML**: Returns the response data as XML (if available and parsed).

**Common Methods:**

- **open(method, url, async)**: Initializes a request with the HTTP method (GET, POST, etc.), target URL, and a boolean indicating whether the request is asynchronous.
- **send(data)**: Sends the request to the server. Data can be included in POST requests.
- **setRequestHeader(header, value)**: Sets HTTP headers before sending the request.
- **abort()**: Cancels an ongoing request.
- **onreadystatechange**: An event handler that is called every time the readyState changes. Commonly used to check when the response is ready and handle it accordingly.

**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.

**Code :**

**script .js**

```
const colleges = [
    "Vivekanand Education Society's Institute of Technology (VESIT)",
    "Vivekanand Education Society's Institute of Management Studies and Research (VESIM)",
    "Vivekanand Education Society's College of Pharmacy (VESCOP)",
    "Vivekanand Education Society's College of Arts, Science and Commerce (VESCOASC)",
    "Vivekanand Education Society's College of Law (VESCOL)",]
const existingUsernames = ['user1', 'user2', 'admin'];
document.addEventListener('DOMContentLoaded', function() {
    const form = document.getElementById('registrationForm');
    const collegeInput = document.getElementById('college');
    const suggestionsDiv =
    document.getElementById('collegeSuggestions'); const messageDiv =
    document.getElementById('registrationMessage');
 collegeInput.addEventListener('input', function()
        { const input = this.value.toLowerCase();
        if (input.length < 2) {
            suggestionsDiv.style.display =
            'none'; return;
        }
 const filteredColleges = colleges.filter(college =>
            college.toLowerCase().includes(input)
        ) if (filteredColleges.length > 0) {
            suggestionsDiv.innerHTML =
            filteredColleges
```

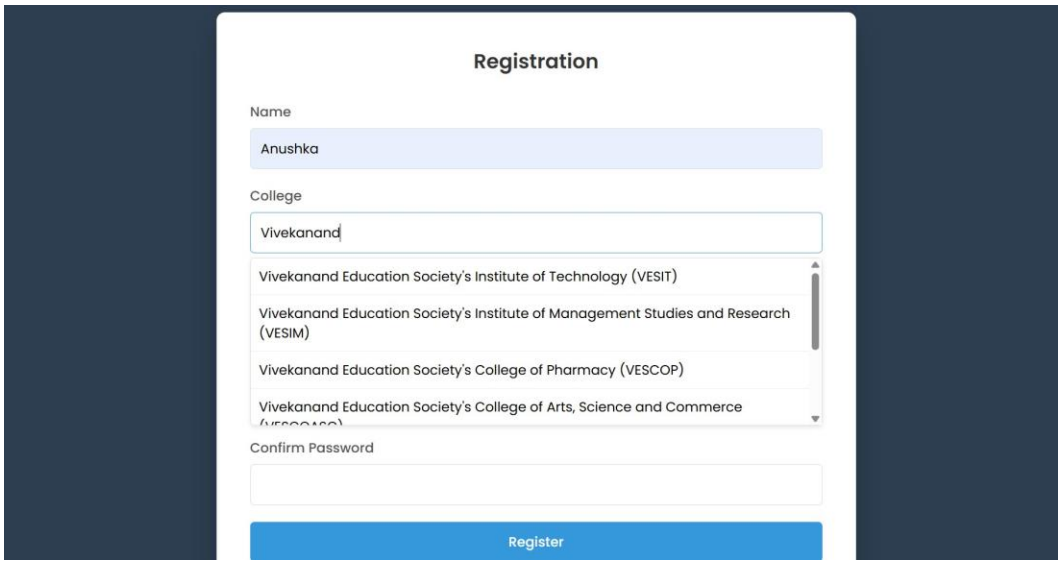
```
            .map(college => `<div>${college}</div>`)
            .join('');
        suggestionsDiv.style.display =
        'block';
    } else {
        suggestionsDiv.style.display = 'none';
suggestionsDiv.addEventListener('click',
function(e) {
    if (e.target.tagName === 'DIV') {
        collegeInput.value = e.target.textContent;
        suggestionsDiv.style.display = 'none';
    }
  }); document.addEventListener('click',
    function(e) { if (e.target !== collegeInput) {
        suggestionsDiv.style.display = 'none';
     }
  });
 form.addEventListener('submit',
    function(e) { e.preventDefault();
    messageDiv.className = 'message';
    messageDiv.textContent = '';
    const name = document.getElementById('name').value.trim();
    const username =
    document.getElementById('username').value.trim(); const
    college = document.getElementById('college').value.trim();
    const password = document.getElementById('password').value;
    const confirmPassword = document.getElementById('confirmPassword').value;
```

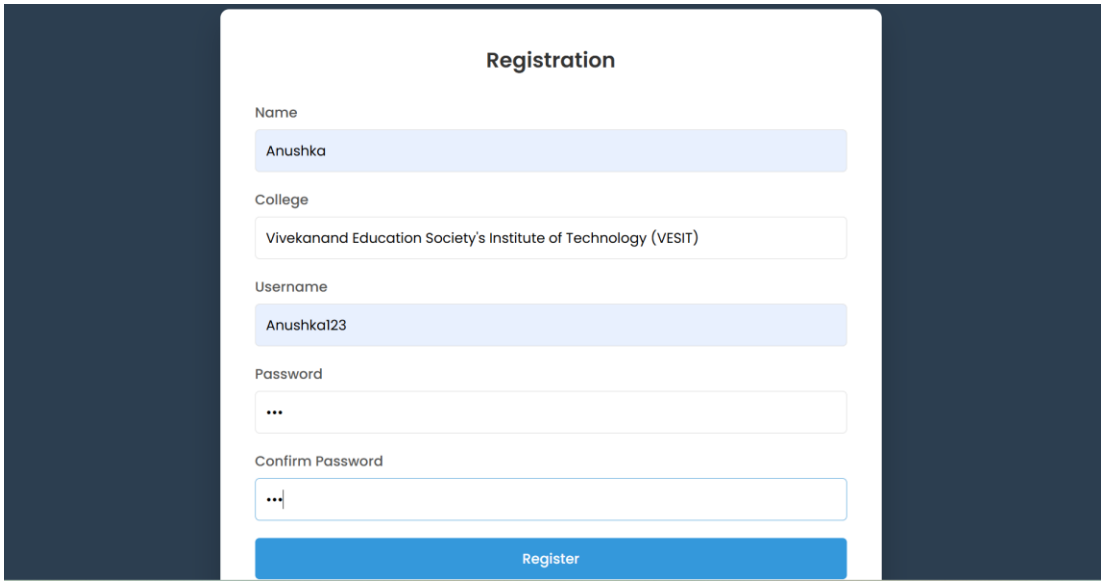
```
        let isValid = true;
    if (name === '') {
            document.getElementById('nameError').textContent = 'Name is required';
            isValid = false;
        } else {
            document.getElementById('nameError').textContent = '';
        }
        if (college === '') {
            document.getElementById('collegeError').textContent = 'College is
            required'; isValid = false;
        } else {
            document.getElementById('collegeError').textContent = '';
        }
        if (password !== confirmPassword) {
            document.getElementById('confirmPasswordError').textContent = 'Passwords do not
            match'; isValid = false;
        } else {
            document.getElementById('confirmPasswordError').textContent = '';
        }

        if (isValid) {
            const data = {
                name,
```

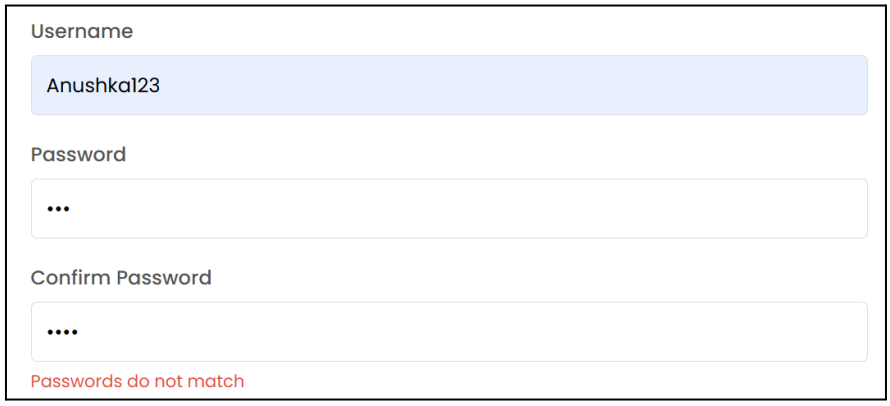
```
    username,
    college,
    password
};

// Create XMLHttpRequest
const xhr = new XMLHttpRequest();
xhr.open('POST', 'http://localhost:3000/register',
true);
xhr.setRequestHeader('Content-Type', 'application/json');

xhr.onload = function() {
    try {
        const response =
        JSON.parse(xhr.responseText); if
        (xhr.status === 200 && response.success)
        {
            messageDiv.className = 'message
            success'; messageDiv.textContent =
            response.message; form.reset();
        } else {
            messageDiv.className = 'message error';
            messageDiv.textContent = response.message || 'Registration failed. Please try again.';
        }
    } catch (error) {
        messageDiv.className = 'message error';
        messageDiv.textContent = 'Error processing response. Please try again.';
```

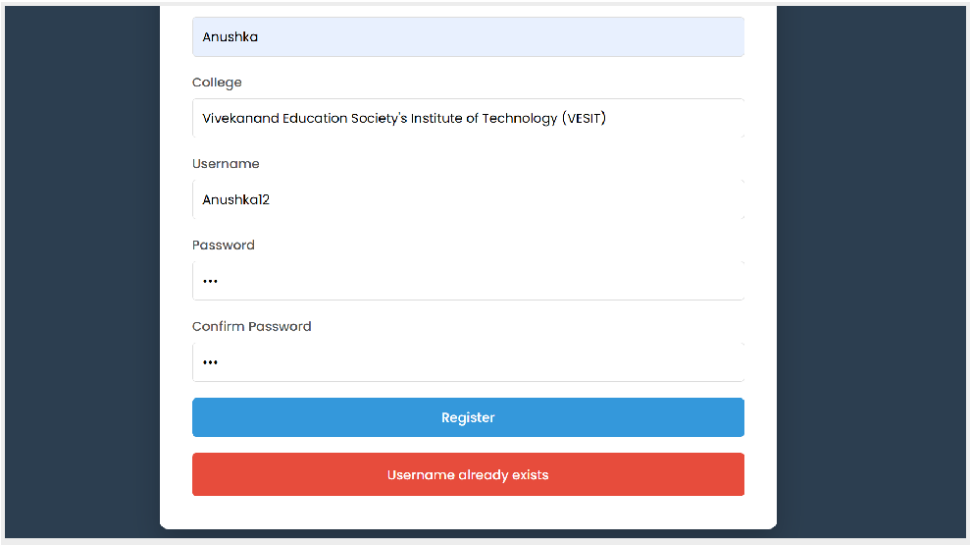
```
            }
        };

        xhr.onerror = function() {
            messageDiv.className = 'message
            error';
            messageDiv.textContent = 'An error occurred. Please try again.';
        };

        xhr.send(JSON.stringify(data));
    }
  });
});
```

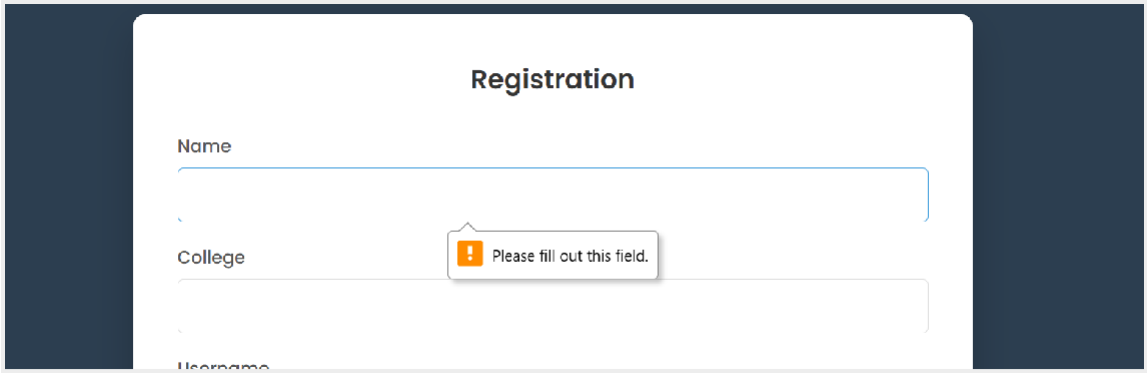
**Output:**

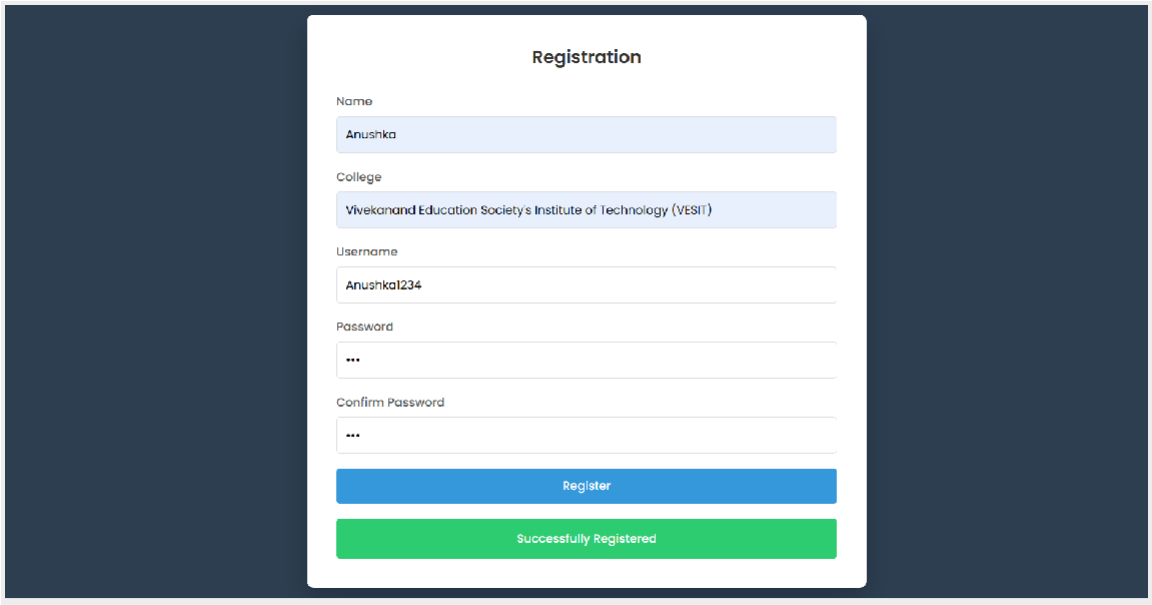



Registration
Name
Anushka
College
Vivekanand Education Society's Institute of Technology (VESIT)
Username
Anushka123
Password
Confirm Password
Register


Username
Anushka123
Password
Confirm Password
Passwords do not match


Anushka
College
Vivekanand Education Society's Institute of Technology (VESIT)
Username
Anushka12
Password
Confirm Password
Register
Username already exists




**Github link :** https://github.com/Skailaje /Experiment_9_WebX

**Conclusion :** In conclusion, understanding the difference between synchronous and asynchronous requests is crucial for creating responsive web applications. The XMLHttpRequest object plays a vital role in enabling asynchronous communication between the client and server without reloading the page. Its properties and methods allow developers to send, receive, and handle data efficiently, improving user experience and performance.