Name : **Sonam Chhabaidiya**

Class : **D15A**

Roll no. : **09**

Experiment – 9: AJAX

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

A) How do Synchronous and Asynchronous Requests differ?

1. Synchronous Requests:

- **Definition:** In a synchronous request, the program waits for the request to complete before continuing with the rest of the code.
- **Behavior:** The process is **blocked**; the code execution is paused until the request is finished, making the system wait for a response.
- **Example:** Making a request to a server for data, and the program doesn't do anything else until the response is received.
- **Impact:** Can cause delays and make the application unresponsive, especially for long-running tasks.

Example in code:

```
var xhr = new XMLHttpRequest();
 xhr.open("GET", "data.json", false);  // 'false' makes the request synchronous
xhr.send();
console.log(xhr.responseText);  // This will only run after the response is received
```

2. Asynchronous Requests:

- **Definition:** In an asynchronous request, the program continues executing other code while waiting for the request to complete.

- **Behavior:** The process is **non-blocking**; the code can perform other tasks while the request is being processed in the background.
- **Example:** A user can continue interacting with the website while data is being fetched from a server.
- **Impact:** Provides a smoother and more responsive user experience because the page doesn't freeze during the request.

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

The XMLHttpRequest (XHR) object is used in web development to send HTTP requests and receive responses asynchronously (without refreshing the web page). It is commonly used for making AJAX requests to interact with web servers and handle data exchange without page reloads.

Here's an overview of the various properties and methods of the XMLHttpRequest object:

Properties of XMLHttpRequest:

1. **`readyState`**
   Description: Represents the state of the request. It returns an integer value (from 0 to 4), indicating the current status of the request.

   - `0`: Request not initialized
   - 1: Server connection established
   - 2: Request received
   - 3: Processing request (response is being downloaded)
   - 4: Request finished and response is ready

   Usage: Used to check the progress of the request.

`2.status`

Description: Holds the HTTP status code of the response received from the server.

- `200`: OK (Request was successful)
- `404`: Not Found (The resource is not available)
- `500`: Internal Server Error

Usage: Used to check if the request was successful and if the server responded properly.

3.`statusText`

Description: Contains the HTTP status message corresponding to the `status` code.

- Example: `"OK"` for `200`, `"Not Found"` for `404`.

Usage: Used for error handling, to show a more user-friendly message about the status of the request.

4.`responseText`

Description: Contains the response body as a string (typically the content returned from the server).

Usage: Used when the response is plain text or XML.

5.`responseXML`

Description: Contains the response body as an XML document (if the server sends XML as the response).

Usage: Used when the response is XML, allowing you to easily parse and manipulate XML data.

6. `responseType`

Description: Defines the type of response expected from the server.

- Values: `""` (default, text), `"json"`, `"document"`, `"blob"`, `"arraybuffer"`.

Usage: Determines how the response should be handled and parsed.

7. `timeout`

Description: Specifies the time (in milliseconds) before the request times out. If the request takes longer than this time, it is aborted.

Usage: Helps to avoid hanging requests and ensures the user doesn't have to wait indefinitely.

8. `withCredentials`

Description: A boolean property that indicates whether or not cross-site requests should include credentials (such as cookies, authorization headers, or TLS client certificates).

Usage: Used for handling cross-origin requests that require authentication.

Methods of XMLHttpRequest:

1. `open(method, url, async, user, password)`

Description: Initializes the request. It must be called before sending the request.

- `method`: HTTP method (`GET`, `POST`, `PUT`, etc.)
- `url`: The URL to send the request to
- `async`: Boolean, `true` (asynchronous) or `false` (synchronous)
- `user` and `password`: Optional, for basic authentication

Usage: Used to specify the type of request (GET, POST, etc.), the target URL, and whether the request is asynchronous or synchronous.

2.`send(data)`

Description: Sends the request to the server. If the method is `POST` or `PUT`, you can send data (such as form data) with the request.

- `data`: The data to be sent to the server (for methods like `POST` or `PUT`). For `GET`, no data is required.

Usage: Used to actually send the request to the server after it has been set up using `open()`.

3.`setRequestHeader(header, value)`

Description: Sets the value of an HTTP request header. This must be done after calling `open()` and before calling `send()`.

Usage: Used to set custom headers, such as `Content-Type`, `Authorization`, etc.

4.`getResponseHeader(header)`

Description: Returns the value of a specific HTTP header from the server's response.

Usage: Used to retrieve individual headers (e.g., `Content-Type`) from the server's response.

5.`getAllResponseHeaders()`

Description: Returns all the response headers as a string.

Usage: Used when you need to access all headers sent back by the server.

6.`abort()`

Description: Aborts the request if it is still in progress.

Usage: Used when you need to stop a request (e.g., if the user cancels an operation).

7.`onreadystatechange`

Description: An event handler that is triggered every time the `readyState` property changes.

Usage: Used to monitor and handle the state changes of the request, often in combination with the `readyState` property.

8.`onload`

Description: An event handler that is triggered when the request is completed successfully (i.e., when `readyState` is 4 and `status` is `200`).

Usage: Used to process the response when the request completes successfully.

9.`onerror`

Description: An event handler that is triggered when an error occurs during the request.

Usage: Used to handle network errors or issues with the request.

10.`ontimeout`

Description: An event handler triggered when the request exceeds the timeout limit (if set with the `timeout` property).

Usage: Used to handle cases when the request takes too long to complete.

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.

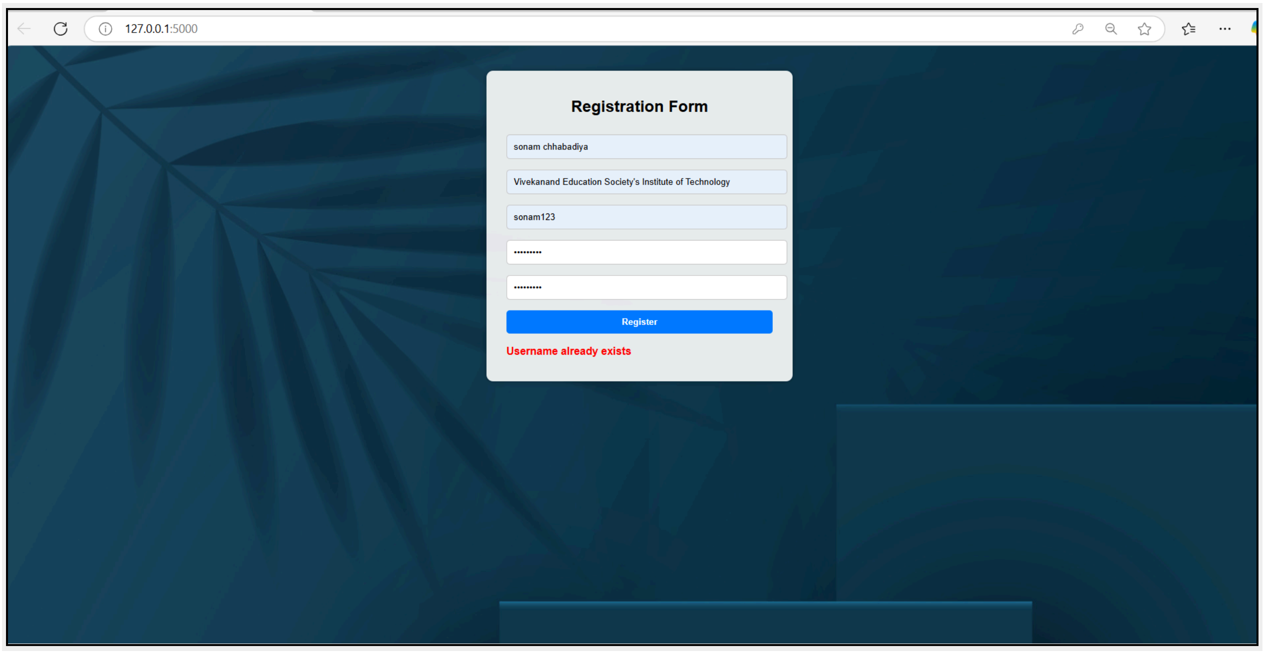
4) Output:

Github link : https://github.com/sonamcc/webxexp9

Screenshots:

1. Form Validation (Name, Username, Passwords):

The registration form performs client-side validation, ensuring the name is not empty, the username is unique, and passwords match before submission.



127.0.0.1:5000

Registration Form

sonam chhabadiya

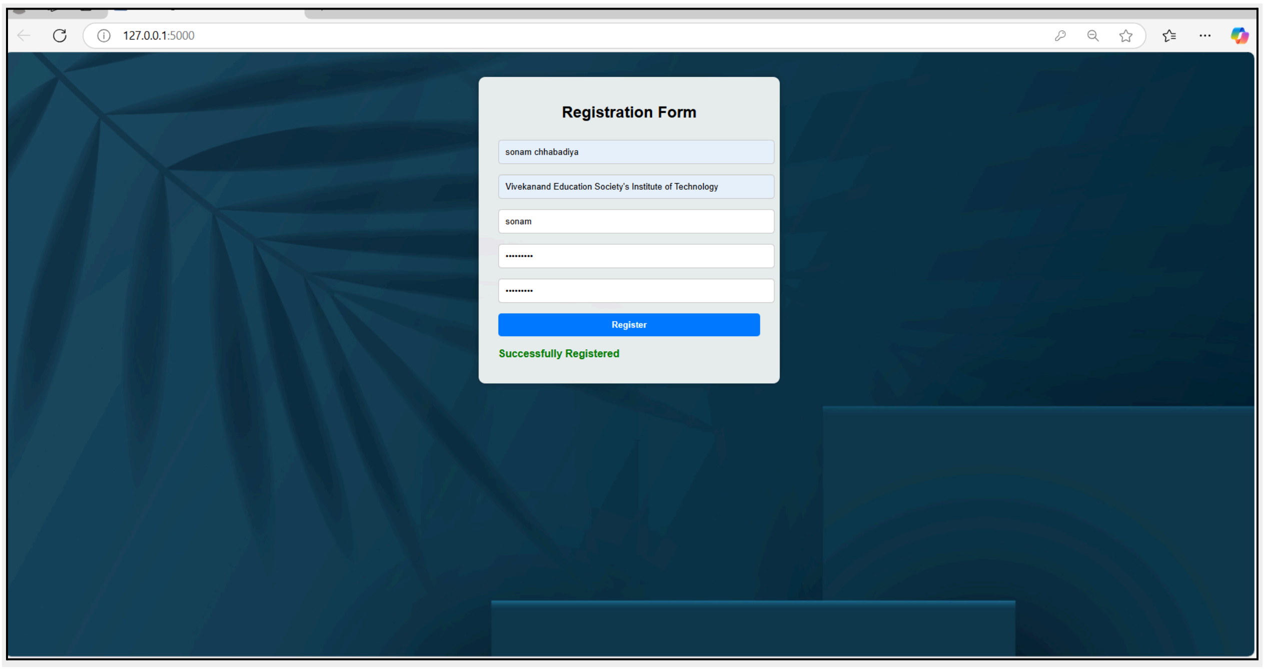
Vivekanand Education Society's Institute of Technology

sonam123

•••••••••

•••••••••

Username already exists



127.0.0.1:5000

Registration Form

sonam chhabadiya

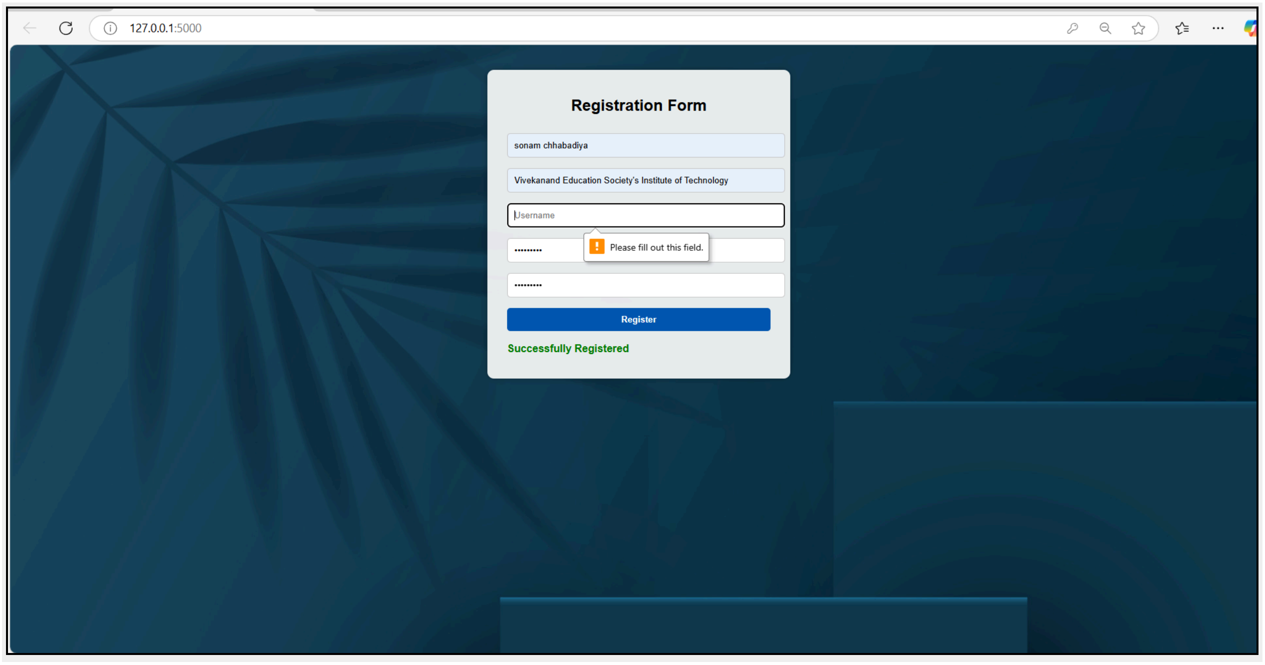
Vivekanand Education Society's Institute of Technology

sonam

•••••••••

•••••••••

Successfully Registered


127.0.0.1:5000

Registration Form

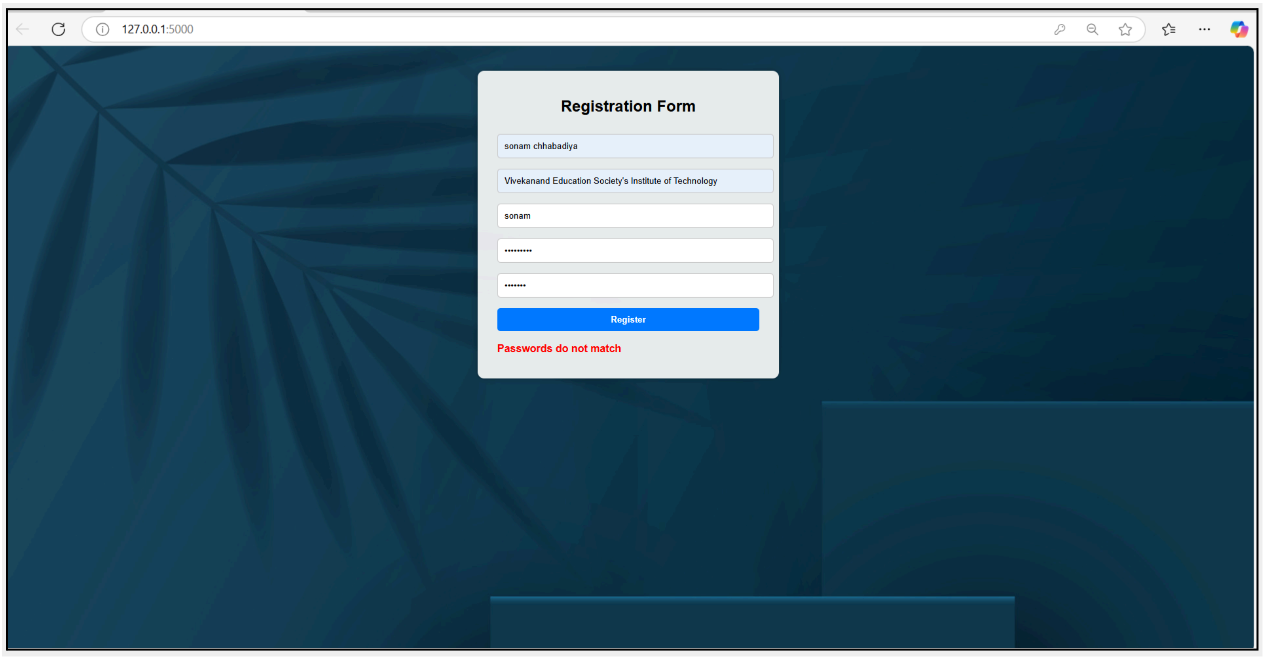
sonam chhabadiya

Vivekanand Education Society's Institute of Technology

Username

Please fill out this field.

Successfully Registered


127.0.0.1:5000

Registration Form

sonam chhabadiya

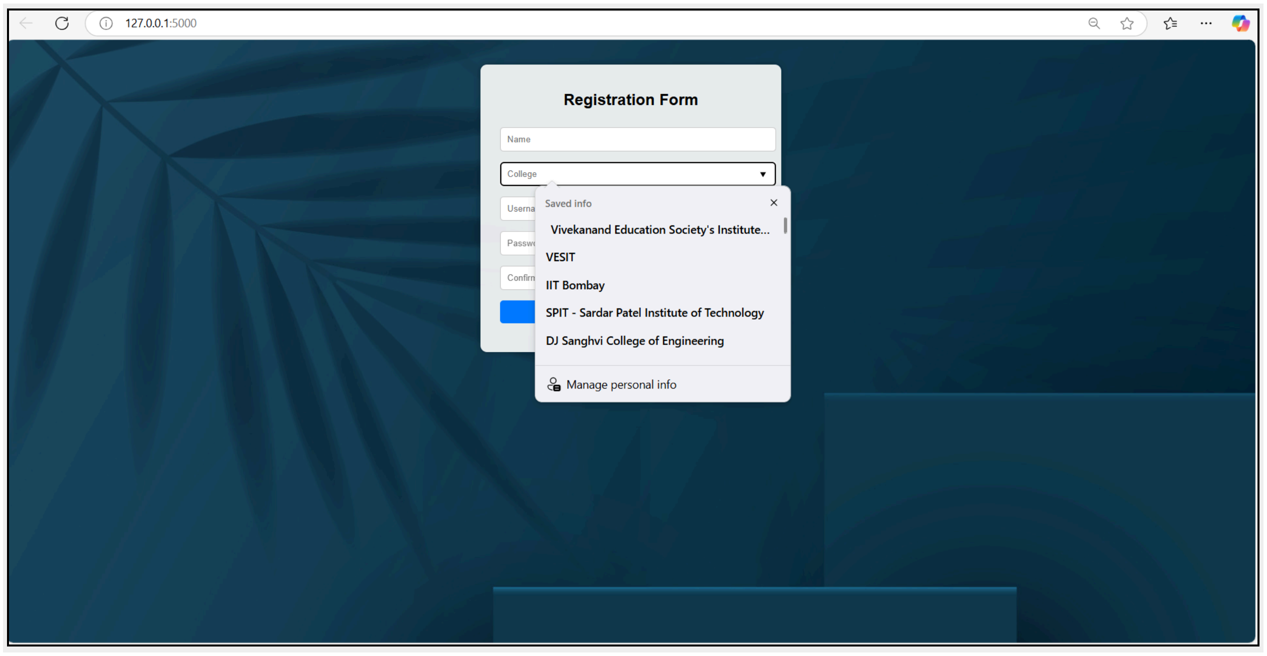
Vivekanand Education Society's Institute of Technology

sonam

Passwords do not match

2.College Name Suggestions (Datalist):

"The form features an autosuggest dropdown for college names, fetched asynchronously from a JSON file using XMLHttpReq*uest."*


127.0.0.1:5000
Registration Form
Name
College
Userna
Passw
Confirn
Saved info
Vivekanand Education Society's Institute...
VESIT
IIT Bombay
SPIT - Sardar Patel Institute of Technology
DJ Sanghvi College of Engineering
Manage personal info

<u>Conclusion:</u>

In this experiment, we implemented a dynamic registration form using AJAX with_the XMLHttpRequest object, where all actions—such as validating non-empty name fields, ensuring unique usernames, matching passwords, and autosuggesting college names from a JSON file—were handled asynchronously without reloading the page. This demonstrated how AJAX enhances user experience by enabling real-time communication with the server and smooth updates to the interface, making web applications more responsive and interactive.