**Name : Chetan Pilane**

**Class :** D15A

**Roll no. :** 44

## Experiment – 6: AJAX

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

A. How do Synchronous and Asynchronous Requests differ?

Synchronous vs. Asynchronous Requests:

1. Synchronous Requests:

○ Blocking Behaviour:

■ Synchronous requests prevent the DOM (Document Object Model) or browser from executing additional code until the server responds.

■ The execution is blocked, and the browser waits for the response.

○ Single Request at a Time:

■ You cannot make another request until you receive the response to the previous one.

○ Usage:

■ Synchronous requests are less common due to their blocking nature.

■ Use the async: false parameter in the AJAX request to make it synchronous.

Example:

```
$.ajax({
url: "/path",
type: "GET",
async: false, // Synchronous request
success: function (response) {
// Handle the response
}
});
```

2. Asynchronous Requests:

○ Non-Blocking Behavior:

■ Asynchronous requests do not wait for the response.

■ The browser continues executing other tasks while waiting for the server's reply.

○ Multiple Requests Simultaneously:
■ You can execute multiple asynchronous requests concurrently.
○ Usage:
■ Asynchronous requests are more common and recommended.
■ Use the async: true parameter (or omit it, as it defaults to true) for asynchronous behavior.

Example:

```
$.ajax({
url: "/path",
type: "GET",
async: true, // Asynchronous request
success: function (response) {
// Handle the response
}
});
```

B. Describe various properties and methods used in XMLHttpRequest Object

Properties and Methods of XMLHttpRequest Object:

1. Properties:
○ onload: Defines a function to be called when the request is received (loaded).
○ onreadystatechange: A function called whenever the readyState property changes.
○ readyState: Represents the current state of the request (0 to 4). Common states:
■ 0: Request not initialized.
■ 1: Server connection established.
■ 2: Request received.
■ 3: Processing the request.
■ 4: Response ready.
○ responseText: Contains the response data as a string.
○ responseXML: Contains the response data as XML.
○ status: Returns the HTTP status code (e.g., 200 for OK, 404 for NOT FOUND).
○ statusText: Returns the status text (e.g., “OK” or “NOT FOUND”).
2. Methods:
○ new XMLHttpRequest(): Creates a new XMLHttpRequest object.
○ abort(): Cancels the current request.
○ getAllResponseHeaders(): Returns HTTP headers as a string.

## 3) **Problem Statement:**

Create a registration page having fields like Name, College, Username and Password (read password twice).

1.index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<title>Registration Form</title>
<link rel="stylesheet" href="style.css">

        Name: Chetan Pilane D15A Roll No. 44
</head>
<body>
<h1>Registration DETAILS</h1>
<form>
<div>
<label for="name">Name:</label>
<input type="text" id="name" name="name">
<span id="name-error"></span>
</div>
<div>
<label for="college">College:</label>
<input type="text" id="college" name="college"
placeholder="Enter your college" list="colleges"
required>
<datalist id="colleges">
<option value="VESIT"></option>
<option value="VIT"></option>
<option value="TSEC"></option>
<option value="KJSCE"></option>
<option value="KJSIEIT"></option>
<option value="FRCRCOE"></option>
<option value="VASANTDADA PATIL COE"></option>
<option value="TERNA COE"></option>
</datalist>
<span id="college-error"></span>
</div>
<div>
<label for="username">Username:</label>
<input type="text" id="username" name="username">
<span id="username-error"></span>
</div>
```

```
<div>
<label for="password">Password:</label>

<input type="password" id="password"

name="password">
<span id="password-error"></span>
</div>
<div>

<label for="confirm-password">RetypePassword:</label>

<input type="password" id="confirm-password" name="confirm-password">
<span id="confirm-password-error"></span>
</div>
<div>
   <button type="submit">Register</button>
</div>
<div>
<span id="registration-message"></span>
</div>
</form>
<script src="script.js"></script>
</body>
</html>
```

2.styles.css

```
body {
   font-family: Arial, sans-serif;
   background-color:greenyellow;
   color: #333;
   }

   h1 {
   font-size: 28px;
   text-align: center;
   margin-top: 50px;
   color: aqua;
   }

   form {
      max-width: 500px;
margin: 0 auto;
padding: 20px;
background-color: #fff;
```

```
border: 1px solid #ddd;
border-radius: 5px;
}
form label {
display: block;
margin-bottom: 5px;
font-weight: bold;
}
form input[type="text"],
form input[type="password"] {
width: 100%;
padding: 8px;
border: 1px solid #ddd;
border-radius: 5px;
margin-bottom: 10px;
box-sizing: border-box;
}
form button[type="submit"] {
display: block;
width: 100%;
background-color: #4CAF50;
color: #fff;
border: none;
border-radius: 5px;
padding: 10px;
cursor: pointer;
}
form button[type="submit"]:hover {
background-color: #3e8e41;
}
#registration-message {
text-align: center;
margin-top: 20px;
font-weight: bold;
color: #4CAF50;
}
```

3.script.js

```
// Get the form element from the HTML document
const form = document.querySelector('form');

// Get the input fields from the form
const nameField = form.querySelector('#name');
const collegeField = form.querySelector('#college');
const usernameField = form.querySelector('#username');
```

```
const passwordField = form.querySelector('#password');
const confirmPasswordField =
form.querySelector('#confirm-password');
const nameError = form.querySelector('#name-error');
const collegeError =
form.querySelector('#college-error');
const usernameError =
form.querySelector('#username-error');
const passwordError =
form.querySelector('#password-error');
const confirmPasswordError =
form.querySelector('#confirm-password-error');
const registrationMessage =
form.querySelector('#registration-message');

// Initialize the users array to store registered users in local storage
let users = JSON.parse(localStorage.getItem('users')) ||
[];

// Function to check if a username is already registered
function isUsernameTaken(username) {
return users.some(user => user.username === username);
}

// Function to add a new user to the users array
function addUser(user) {
   users.push(user);
localStorage.setItem('users', JSON.stringify(users));
}

// Function to check password strength
function isPasswordStrong(password) {
// Minimum length of 8 characters
// At least one uppercase letter, one lowercase letter,and one number

const passwordRegex =/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$/;
return passwordRegex.test(password);
}

// Function to hash the password using SHA-256
function hashPassword(password) {

return crypto.subtle.digest('SHA-256', new
```

```
TextEncoder().encode(password))
.then(buffer => {
return Array.from(new Uint8Array(buffer))
.map(b => b.toString(16).padStart(2, '0'))
.join('');
});
}

// Handle form submission
form.addEventListener('submit', handleFormSubmit);

function handleFormSubmit(event) {
event.preventDefault();

// Reset the error messages
nameError.textContent = '';
collegeError.textContent = '';
usernameError.textContent = '';
passwordError.textContent = '';
confirmPasswordError.textContent = '';
registrationMessage.textContent = '';

// Get the form values
const name = nameField.value.trim();
const college = collegeField.value.trim();
const username = usernameField.value.trim();
const password = passwordField.value;
const confirmPassword = confirmPasswordField.value;
let isValid = true;

// Validate the name field
if (name === '') {
nameError.textContent = 'Please enter your name';
isValid = false;
}

// Validate the college field
if (college === '') {

collegeError.textContent = 'Please enter your college';
isValid = false;
}

// Validate the username field
```

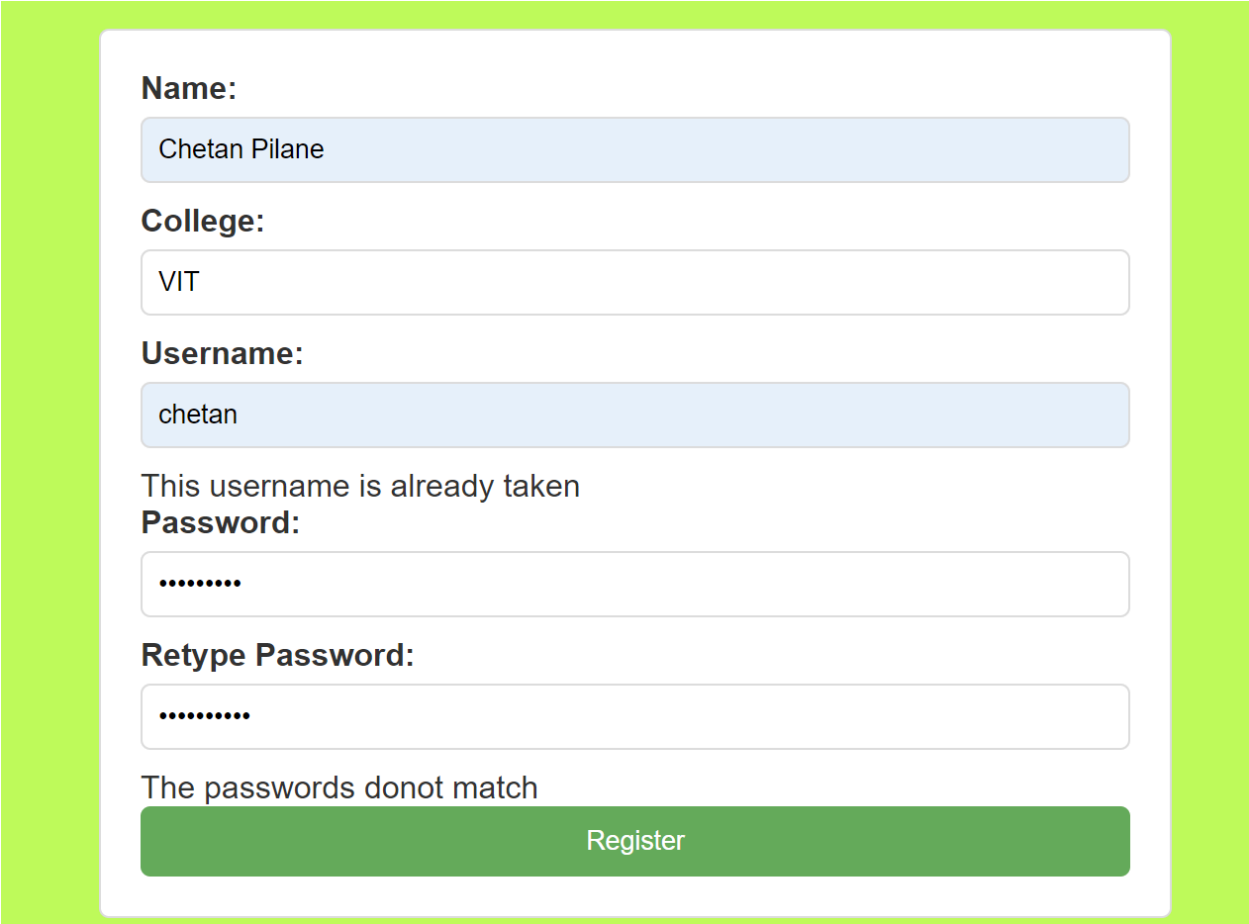
```
if (username === '') {

usernameError.textContent = 'Please enter a username';
isValid = false;
} else if (isUsernameTaken(username)) {
   usernameError.textContent = 'This username is already taken';
isValid = false;
}

// Validate the password fields
if (password === '') {

passwordError.textContent = 'Please enter apassword';
isValid = false;
} else if (!isPasswordStrong(password)) {
passwordError.textContent = 'Password must be atleast 8 characters long and contain at least oneuppercase letter, one lowercase letter, and one number';
isValid = false;
}
if (confirmPassword === '') {
confirmPasswordError.textContent = 'Please retypeyour password';
isValid = false;
}
if (password !== confirmPassword) {
confirmPasswordError.textContent = 'The passwords donot match';
isValid = false;
}

if (isValid) {
// Create a new user object
hashPassword(password).then(hashedPassword => {
const user = {
name,
college,
username,
password: hashedPassword
};

// Add the new user to the users array and save itto local storage
addUser(user);

// Show the success message
registrationMessage.textContent = 'Successfullyregistered!';
});
```

```
}
}
4.server.js
const http = require('http');
const fs = require('fs');
const path = require('path');

const server = http.createServer((req, res) => {
if (req.url === '/') {
fs.readFile(path.join(__dirname, 'index.html'), (err,
data) => {
if (err) {
res.statusCode = 500;
res.end('Error loading index.html');
} else {
res.setHeader('Content-Type', 'text/html');
res.end(data);
}
});
} else if (req.url === '/script.js') {
fs.readFile(path.join(__dirname, 'script.js'), (err,
data) => {
   if (err) {
      res.statusCode = 500;
      res.end('Error loading script.js');
      } else {

      res.setHeader('Content-Type',

      'application/javascript');
      res.end(data);
      }
      });
      } else if (req.url === '/register' && req.method ===
      'POST') {
      let body = '';
      req.on('data', (chunk) => {
      body += chunk.toString();
      });
      req.on('end', () => {
      console.log('Received registration data:', body);
      res.setHeader('Content-Type', 'application/json');
      res.end(JSON.stringify({ message: 'Registrationsuccessful' }));
      });
```

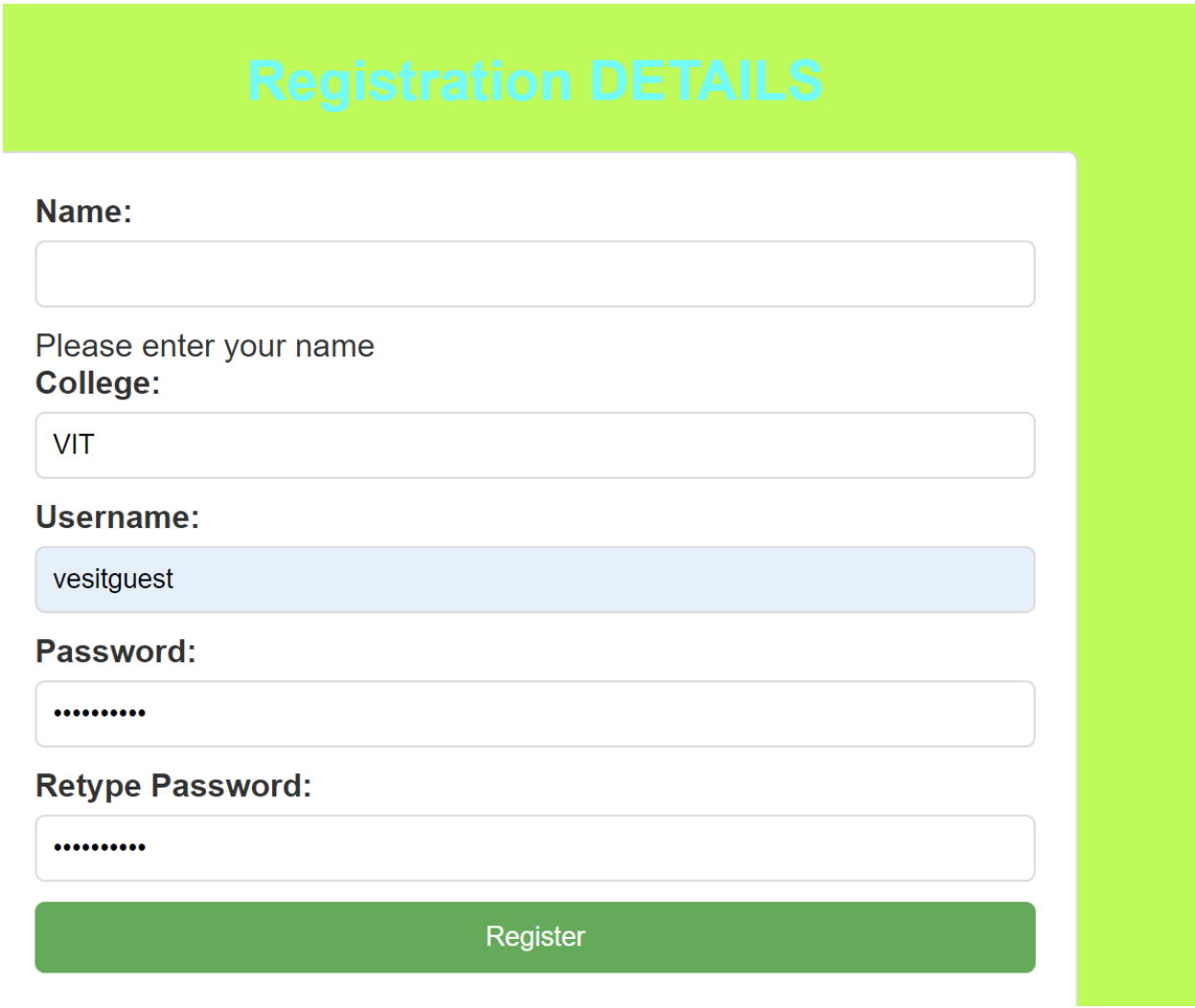
```
        } else {
        res.statusCode = 404;
        res.end('Not found');
}
});

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
console.log(`Server is running on port ${PORT}`);
});
```

Validate the form by checking for

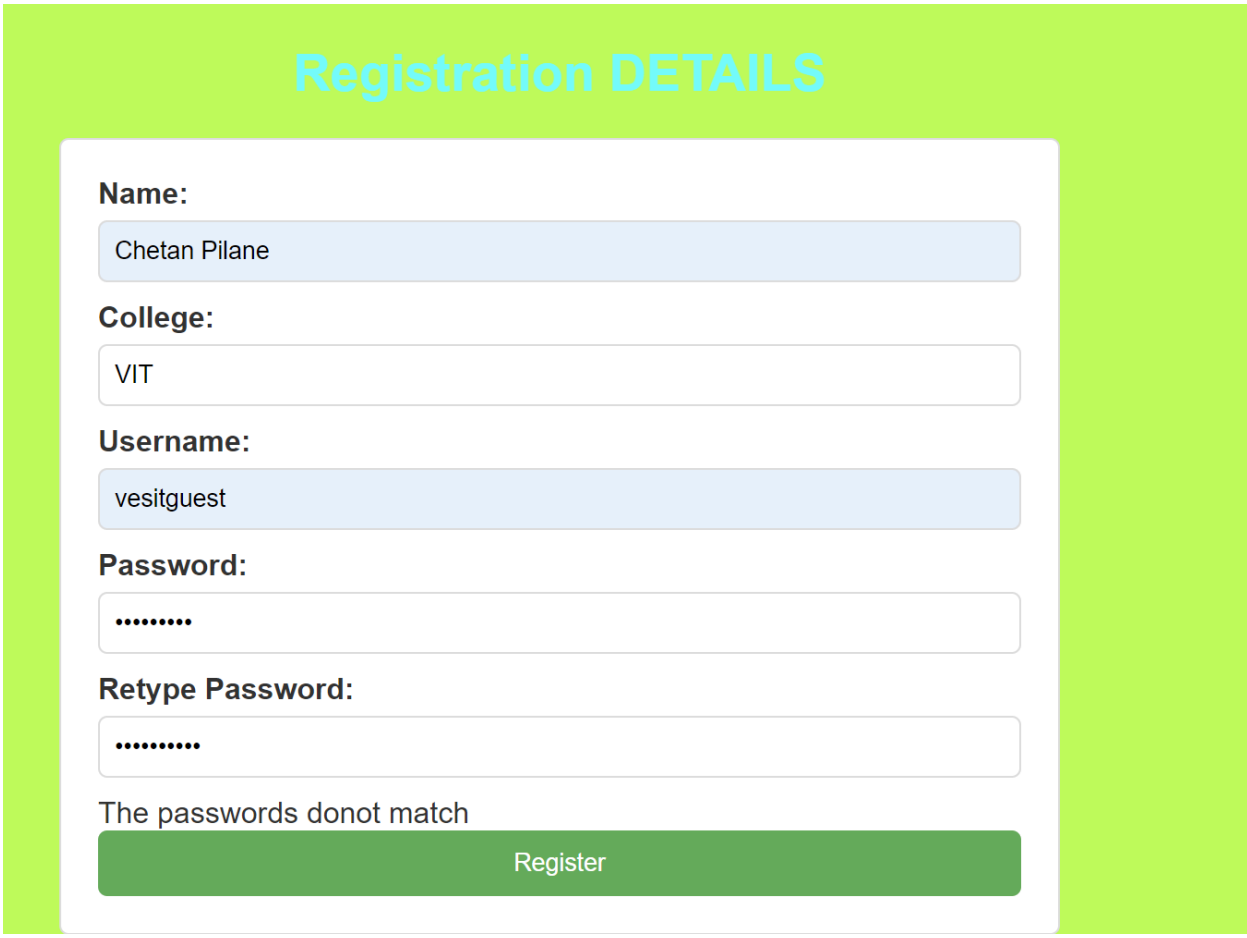
1. Usernameis not same as existing entries



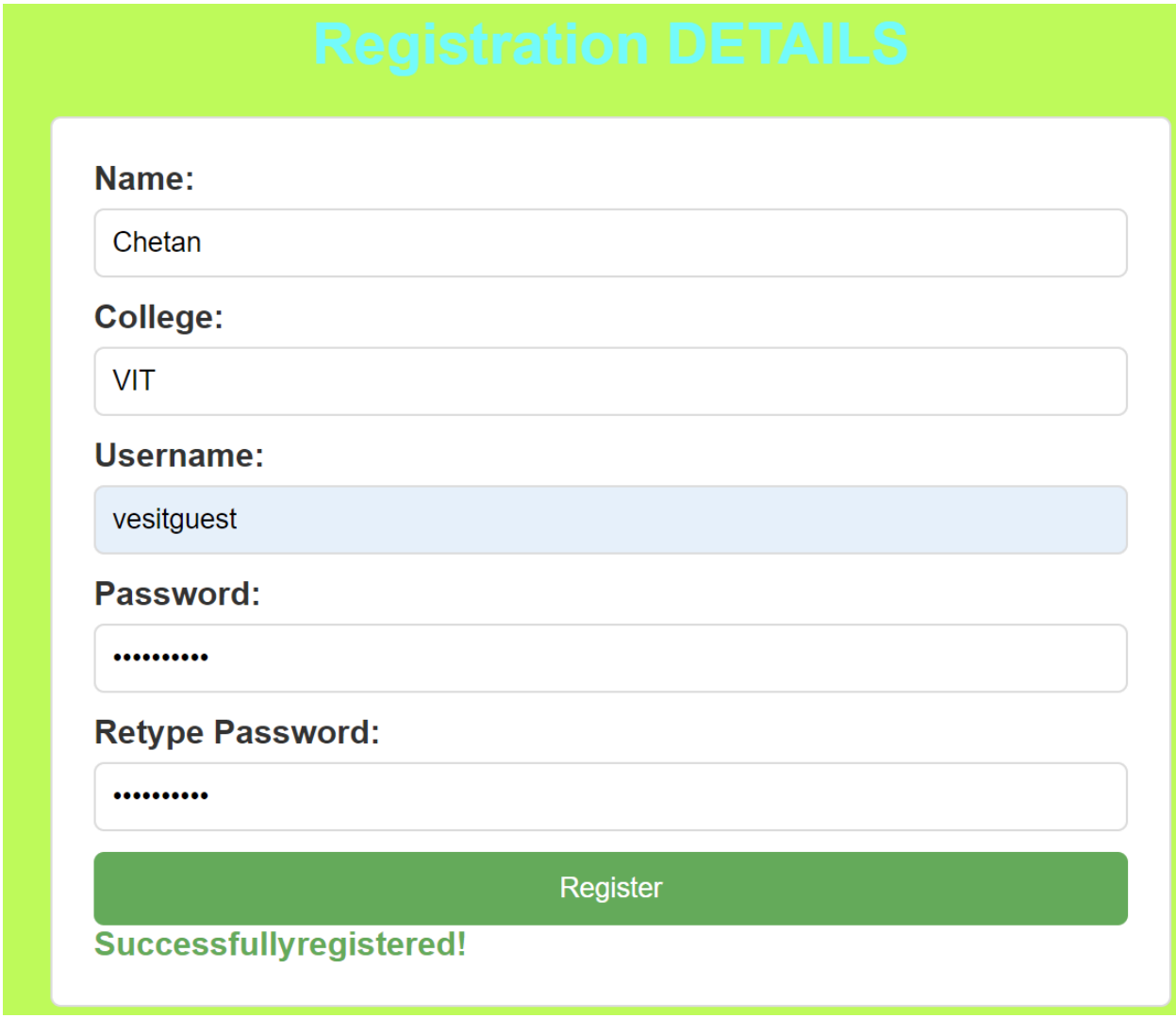

2. Name field is not empty




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.




**Conclusion: Studying AJAX is essential for enabling asynchronous server communication and creating dynamic, responsive web applications.**