FORM VALIDATION

The code is for a client-side form validation script that ensures user input meets specific criteria before submission. The form includes fields for name, email, password, password confirmation, and date of birth. If all validations pass, the form redirects the user to a specified URL; otherwise, it alerts the user to fix errors.

1. **Event Listener for DOMContentLoaded**

   Ensures the JavaScript code runs only after the entire DOM has been fully loaded. This prevents any attempts to manipulate elements that may not yet be available in the document.

   document.addEventListener('DOMContentLoaded', () => {

2. **Element References**

   const form = document.getElementById('registrationForm');

   const nameField = document.getElementById('name');

   const emailField = document.getElementById('email');

   const passwordField = document.getElementById('password');

   const confirmPasswordField = document.getElementById('confirmPassword');

   const dobField = document.getElementById('dob');

   const submitBtn = document.getElementById('submitBtn');

   const nameError = document.getElementById('nameError');

   const emailError = document.getElementById('emailError');

   const passwordError = document.getElementById('passwordError');

   const confirmPasswordError = document.getElementById('confirmPasswordError');

   const dobError = document.getElementById('dobError');

   Retrieves references to the form and its input fields, as well as error message elements, using their id attributes. These references are used later for validation and updating the UI.

3. **Validation Functions**

```
const isValidName = (name) => /^[A-Za-z\s]{3,}$/.test(name);

const isValidEmail = (email) => /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);

const isValidPassword = (password) => /^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/.test(password);

const isValidDob = (dob) => {
    const today = new Date();
    const birthDate = new Date(dob);
    let age = today.getFullYear() - birthDate.getFullYear();
    const monthDiff = today.getMonth() - birthDate.getMonth();
    if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {
        age--;
    }
    return age >= 18;
};
```

isValidName: Validates that the name contains only letters and spaces, and is at least 3 characters long.

isValidEmail: Validates that the email has a proper format (e.g., user@example.com).

isValidPassword: Validates that the password is at least 8 characters long and includes letters, numbers, and special characters.

isValidDob: Checks if the user is at least 18 years old by comparing the current date with the provided birth date.

4. **Field Validation Function**

```
const validateField = (field, validator, errorField) => {
    const isValid = validator(field.value.trim());
    field.classList.toggle('border-red-500', !isValid);
    field.classList.toggle('border-green-500', isValid);
    errorField.classList.toggle('hidden', isValid);
    return isValid;
};
```

Validates a specific field using the provided validator function.

Updates the border color of the field based on validity (border-red-500 for invalid, border-green-500 for valid).

Shows or hides the associated error message based on validity.

5. **Form Validation Function**

```
const validateForm = () => {
   const isNameValid = validateField(nameField, isValidName, nameError);
   const isEmailValid = validateField(emailField, isValidEmail, emailError);
   const isPasswordValid = validateField(passwordField, isValidPassword,
passwordError);
   const isConfirmPasswordValid = validateField(confirmPasswordField,
      (confirmPassword) => confirmPassword === passwordField.value,
confirmPasswordError);
   const isDobValid = validateField(dobField, isValidDob, dobError);

   const isFormValid = isNameValid && isEmailValid && isPasswordValid &&
isConfirmPasswordValid && isDobValid;
   submitBtn.disabled = !isFormValid;
   return isFormValid;
};
```

Calls validateField for each form field to ensure all fields are valid.

Disables the submit button if the form is invalid.

Returns the overall validity of the form.

6. **Event Listeners**

```
form.addEventListener('input', validateForm);

form.addEventListener('submit', (e) => {
   e.preventDefault();
   if (validateForm()) {
      window.location.href = 'https://christuniversity.in';
   } else {
      alert('Please fix the errors in the form.');
   }
});
```

input Event: Triggers validation whenever the user types into any form field.

submit Event: Prevents the default form submission behavior. If the form is valid, redirects the user to the specified URL. If not, shows an alert prompting the user to fix the errors.

Rationale Behind Design and Implementation Decisions

Client-Side Validation:

Purpose: Provides immediate feedback to users, enhancing user experience and preventing invalid data from being sent to the server.

Implementation: JavaScript functions and event listeners check the validity of form fields as the user interacts with the form and before submission.

Dynamic Feedback:

Visual Indicators: Uses Tailwind CSS classes to change field borders and show error messages dynamically based on validation results. This makes it clear to users which fields need correction.

Error Messages: Specific error messages are displayed when validation fails, guiding users on how to correct their input.

Field-Specific Validation:

Separation of Concerns: Each field has its own validation logic, making the code modular and easier to maintain. For example, name validation is independent of email or password validation.

Form Validation on Input and Submit:

Immediate Validation: By validating on input, users receive instant feedback, which helps correct errors in real-time.

Final Validation: On form submission, validation is checked again to ensure all data is correct before redirecting or alerting the user.

Redirection and User Feedback:

Redirect: On successful validation, the user is redirected to a specific URL (https://christuniversity.in), which is suitable for a real-world scenario where users are guided to a confirmation or thank you page.

Alert: If validation fails, an alert informs users to fix the errors, ensuring that they are aware of what needs to be corrected.