Don Bosco Institute of Technology
Kurla (W), Mumbai 400 070

Internet Programming Lab

Name: Umme Atiya Quraishi
Class: TEIT
Roll no.: 47
Date : 30/08/2024
Experiment no. 7
Title: To Create a web pages using JavaScript Validations

Theory:

JavaScript validation refers to the process of checking and verifying the input data in web forms or other input fields to ensure that it meets specific criteria before being processed or sent to the server. This is crucial for enhancing user experience and maintaining data integrity.

Purpose of JavaScript Validations

1. **Improving User Experience**: Providing immediate feedback to users regarding their input helps them correct mistakes before submitting forms, reducing frustration.
2. **Data Integrity**: Validations ensure that only properly formatted data is accepted, which helps maintain the integrity of the data in the application.
3. **Security**: Proper validation can protect against malicious input, such as SQL injection or cross-site scripting (XSS).
4. **Reducing Server Load**: By validating data on the client side, unnecessary server requests can be minimized, improving performance.

Types of Validations

1. **Client-Side Validation**:
    - Conducted in the user's browser before data is sent to the server. It offers immediate feedback and improves user experience.
    - Example: Checking if an email address is in the correct format before submitting the form.
2. **Server-Side Validation**:

    - Conducted on the server after data is sent from the client. This is essential for security and ensuring data integrity.
    - Example: Checking if a username already exists in the database during registration.
3. **Synchronous vs. Asynchronous Validation**:
    - **Synchronous**: Immediate checks that block the execution until completed (e.g., regex validations).
    - **Asynchronous**: Checks that might involve server calls (e.g., validating if a username is available).

Common Validation Techniques

1. **Required Fields**: Ensure that necessary fields are filled out by the user.
    - Example: Name and email fields in a registration form.
2. **Data Type Validation**: Check if the input data is of the expected type (e.g., number, string).
    - Example: Verifying that a phone number contains only digits.
3. **Format Validation**: Use regular expressions to ensure data follows a specific format (e.g., email, date).
    - Example: Validating that an email address follows the pattern `username@domain.com`.
4. **Length Validation**: Check if the input data meets the required length criteria.
    - Example: Password fields must contain at least 8 characters.
5. **Range Validation**: Verify that numeric values fall within a specified range.
    - Example: Age must be between 18 and 65.
6. **Custom Validations**: Implement specific validation logic based on the application's requirements.
    - Example: Validating that a credit card number is a valid format.

Methods for JavaScript Validations

1. **HTML5 Validation Attributes**:
    - Use built-in HTML attributes like `required`, `minlength`, `maxlength`, `pattern`, and `type` to perform basic validations without writing JavaScript.

Example:
html
Copy code

```
<input type="email" required
pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$" />
```

2. **JavaScript Functions**:
    - Create functions to validate input fields manually. This offers greater flexibility and control over the validation process.

Example:
javascript
Copy code

```
function validateEmail(email) {
  const pattern = /^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,}$/;
  return pattern.test(email);
}
```

3. **Form Submission Handling**:
    - Intercept form submissions to perform validations and prevent submission if validation fails.

Example:
javascript
Copy code

```
const form = document.getElementById('myForm');
form.addEventListener('submit', (e) => {
  const email = document.getElementById('email').value;
  if (!validateEmail(email)) {
    e.preventDefault(); // Prevent form submission
    alert('Invalid email format');
  }
});
```

Best Practices for JavaScript Validations

1. **Client-Side & Server-Side Validation**: Always validate data on both client and server sides for enhanced security and integrity.
2. **User-Friendly Messages**: Provide clear, specific, and friendly error messages to guide users in correcting their input.
3. **Regular Expressions**: Use regular expressions judiciously; they can be powerful but may become complex and hard to maintain.
4. **Accessibility**: Ensure that validation messages are accessible to all users, including those using screen readers.
5. **Avoid Over-Validation**: Do not make validation too strict, which may frustrate users. Allow for flexibility where appropriate.
6. **Testing**: Thoroughly test validation logic to cover various scenarios, including edge cases.

Examples of JavaScript Validations

Email Validation:
javascript
Copy code

```
function validateEmail(email) {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}
```

1.

Password Strength Validation:
javascript
Copy code

```
function validatePassword(password) {
  const hasUpperCase = /[A-Z]/.test(password);
  const hasLowerCase = /[a-z]/.test(password);
  const hasNumbers = /\d/.test(password);
  const hasSpecialChars = /[!@#$%^&*]/.test(password);
  const isValidLength = password.length >= 8;

  return hasUpperCase && hasLowerCase && hasNumbers &&
hasSpecialChars && isValidLength;
```

```
}
```

2.

Phone Number Validation:
javascript
Copy code

```
function validatePhoneNumber(phone) {
  const regex = /^\d{10}$/; // Adjust regex as needed for
your format
  return regex.test(phone);
}
```

3.

Form Submission with Validations:
html
Copy code

```
<form id="myForm">
  <input type="text" id="email" placeholder="Enter your
email" required />
  <button type="submit">Submit</button>
</form>
<script>

document.getElementById('myForm').addEventListener('submit',
function (e) {
    const email = document.getElementById('email').value;
    if (!validateEmail(email)) {
      e.preventDefault();
      alert('Please enter a valid email address.');
    }
  });
</script>
```

4.

Conclusion

JavaScript validations play a vital role in web development by ensuring that user input is accurate, secure, and appropriate before processing. By implementing both client-side and server-side validations, developers can enhance the user experience, maintain data integrity, and protect their applications from potential vulnerabilities. Proper validation techniques contribute to building reliable, user-friendly web applications.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript Validation and Array Example</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        input, button {
            padding: 10px;
            margin: 10px 0;
            width: 100%;
        }
        #result, #arrayResult {
            margin-top: 20px;
            color: green;
        }
    </style>
</head>
<body>

    <h1>User Registration Form</h1>

    <!-- Form to collect user details -->
```

```
    <label for="name">Name:</label>
    <input type="text" id="name" placeholder="Enter your name"><br>

    <label for="email">Email:</label>
    <input type="text" id="email" placeholder="Enter your email"><br>

    <label for="dob">Date of Birth:</label>
    <input type="date" id="dob"><br>

    <label for="hobbies">Hobbies (comma separated):</label>
    <input type="text" id="hobbies" placeholder="Enter your hobbies"><br>

    <button onclick="registerUser()">Register</button>

    <p id="result"></p>
    <p id="arrayResult"></p>

    <script>
        // Function to handle user registration
        function registerUser() {
            // Input fields
            let name = document.getElementById("name").value;
            let email = document.getElementById("email").value;
            let dob = document.getElementById("dob").value;
            let hobbies = document.getElementById("hobbies").value;

            // Validate name
            if (name === "" || name.length < 3) {
                document.getElementById("result").innerHTML = "Please enter a valid name
with at least 3 characters.";
                return;
            }

            // Validate email using simple check
            if (!email.includes("@") || !email.includes(".")) {
                document.getElementById("result").innerHTML = "Please enter a valid
email.";
```

```
            return;
        }

        // Validate date of birth
        if (dob === "") {
            document.getElementById("result").innerHTML = "Please enter your date of
birth.";
            return;
        }

        // Validate hobbies - ensuring at least one hobby is entered
        if (hobbies === "") {
            document.getElementById("result").innerHTML = "Please enter your
hobbies.";
            return;
        }

        // Split hobbies into an array (string operation)
        let hobbiesArray = hobbies.split(",").map(hobby => hobby.trim());

        // Check if more than one hobby was entered
        if (hobbiesArray.length < 1) {
            document.getElementById("result").innerHTML = "Please enter at least one
hobby.";
            return;
        }

        // Calculate age based on date of birth
        let age = calculateAge(dob);

        // Display user details
        document.getElementById("result").innerHTML = `Name: ${name}<br>Email:
${email}<br>Age: ${age}<br>Hobbies: ${hobbiesArray.join(", ")}`;

        // Array operation: add a new hobby and display all hobbies
        hobbiesArray.push("New Hobby");
```

```
        document.getElementById("arrayResult").innerHTML = `Updated Hobbies:
${hobbiesArray.join(", ")}`;
    }

    // Function to calculate age based on date of birth
    function calculateAge(dob) {
        let dobDate = new Date(dob);
        let today = new Date();
        let age = today.getFullYear() - dobDate.getFullYear();
        let monthDiff = today.getMonth() - dobDate.getMonth();

        if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < dobDate.getDate()))
{
            age--;
        }

        return age;
    }
  </script>

</body>
</html>
```

Output: