COLLEGE CODE : 9513

COLLEGE NAME : JAYARAJ ANNAPACKIAM CSI COLLEGE OF

ENGINEERING

DEPARTMENT : COMPUTER SCIENCEAND ENGINEERING

STUDENT NM-ID : 6664EC010D7DC70874249A8BF75439FD

ROLL NO : 951323104036

DATE : 04.10.2025

Completed the project named as

Phase1 TECHNOLOGY

PROJECT NAME : Interactive form validation

SUBMITTED BY,

NAME : Pooja M
MOBI LE NO: 9043450025

1. Final Demo Walkthrough: Interactive Form Validation:

❖ **Introduction:**

- Briefly introduce your form:

  “This is an interactive form designed to give instant feedback to users as they fill out their details. It ensures data correctness before submission, improving user experience.”

❖ **Display the Form:**

- Show the form interface:
  - Fields: Name, Email, Password, Confirm Password, Phone Number.
  - Buttons: Submit, Reset.
- Mention that the form is **responsive** and works in real-time.

❖ **Show Real-Time Validation:**

- **Name Field:**
  - Enter invalid input (like numbers or symbols).
  - Highlight how the form shows: *“Name must contain only letters.”*
- **Email Field:**
  - Enter wrong email format (e.g., `abc@com`).
  - Validation shows: *“Please enter a valid email address.”*
- **Password Field:**
  - Show requirements like: min 8 characters, at least 1 number, 1 special character.
  - Typing an invalid password instantly shows which criteria are missing.
- **Confirm Password Field:**
  - Enter a password that doesn’t match.
  - Show error: *“Passwords do not match.”*
- **Phone Number Field:**
  - Enter letters or too few digits.
  - Show error: *“Please enter a valid 10-digit phone number.”*

❖ **Interactive Feedback:**

- Show **success feedback** when fields are correct (green borders, checkmarks, etc.).
- Demonstrate **error messages disappearing automatically** once the input is corrected.

❖ **Form Submission:**

- Try to submit with **errors**:

    - Show that submission is **prevented** and the form highlights remaining errors.
- Correct all inputs and submit:
    - Show **successful submission message** like: *“Form submitted successfully!”*

❖ **Extra Features (Optional for Wow Factor):**

- Real-time **password strength meter**.
- Animated validation icons (🟩 or +) beside fields.
- Tooltips explaining why the input is invalid.
- Disable submit button until all fields are valid.

❖ **Code Highlights (Optional):**

- Show **key JavaScript snippets**:
    - Event listeners for `input` or `blur`.
    - Regex validation for email, phone, password.
    - DOM manipulation for showing errors dynamically.

❖ **Conclusion:**

- Summarize:

    “This interactive form ensures users enter valid data in real-time, prevents errors, and provides an engaging user experience. It’s fully responsive, easy to use, and ready to integrate into any web application.

2. Project Report

Interactive Form Validation:

❖ Title:

Interactive Form Validation Using HTML, CSS, and JavaScript

❖ Abstract:

This project focuses on designing an **interactive web form** that validates user input in **real-time**. The main aim is to improve user experience and ensure data correctness before submission. The form checks for correct formats, such as valid email addresses, password strength, name format, and phone numbers. Instant feedback helps users correct mistakes immediately, reducing errors and enhancing usability.

❖ Introduction:

Forms are an essential part of any web application. Often, users submit forms with incorrect or incomplete information, which leads to errors and poor user experience.
This project implements an **interactive form with validations**, where users get immediate feedback while typing. It uses **HTML** for structure, **CSS** for styling, and **JavaScript** for validations and interactivity.

❖ Objectives:

- To create a user-friendly web form.
- To validate user inputs in **real-time**.
- To prevent invalid form submissions.
- To provide instant feedback for correcting errors.
- To enhance **user experience** and reduce form submission errors.

❖ Tools and Technologies:

- **HTML5** – for creating the structure of the form.
- **CSS3** – for styling, responsive design, and visual feedback.
- **JavaScript** – for real-time validations and interactivity.
- **Optional:** Bootstrap for responsive layout.

❖ Features:

1. **Real-Time Validation**
    - Name: Only letters allowed.
    - Email: Must follow standard email format.
    - Password: Minimum 8 characters, includes numbers and special characters.
    - Confirm Password: Must match the password.
    - Phone Number: Must be 10 digits.
2. **Interactive Feedback**
    - Green check marks for valid fields.
    - Red warning messages for invalid fields.
3. **Prevent Form Submission on Errors**
4. **Password Strength Meter (Optional)**
5. **Responsive Design**

❖ System Design:

a) Flow Diagram:

```
User Opens Form → User Enters Data → JavaScript Validates Each Field
     ↓
```

```
If Field Invalid → Display Error Message
If All Fields Valid → Enable Submit Button → Form Submitted Successfully
```

b) Validation Logic:

- **Name:** /^[A-Za-z\s]+$/
- **Email:** /^[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+.[a-zA-Z]{2,4}$/
- **Password:** Minimum 8 characters, at least 1 number and 1 special character.
- **Confirm Password:** Must match Password.
- **Phone Number:** /^[0-9]{10}$/

❖ Implementation:

HTML Structure Example:

```
<form id="registrationForm">
  <label>Name:</label>
  <input type="text" id="name" required>
  <span class="error" id="nameError"></span>

  <label>Email:</label>
  <input type="email" id="email" required>
  <span class="error" id="emailError"></span>

  <label>Password:</label>
  <input type="password" id="password" required>
  <span class="error" id="passwordError"></span>

  <label>Confirm Password:</label>
  <input type="password" id="confirmPassword" required>
  <span class="error" id="confirmPasswordError"></span>

  <label>Phone Number:</label>
  <input type="text" id="phone" required>
  <span class="error" id="phoneError"></span>

  <button type="submit">Submit</button>
</form>
```

JavaScript Validation Example:

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

name.addEventListener('input', () => {
  const regex = /^[A-Za-z\s]+$/;
  if (!regex.test(name.value)) {
    nameError.textContent = "Name must contain only letters";
  } else {
    nameError.textContent = "";
  }
});
```

CSS Example:

```
.error {
  color: red;
  font-size: 12px;
}
input:valid {
  border: 2px solid green;
}
input:invalid {
  border: 2px solid red;
}
```

❖ Screenshots:

Form Interface

Validation Errors

Successful Submission

❖ Advantages:

- Reduces form submission errors.
- Enhances user experience with immediate feedback.
- Prevents invalid data from entering the system.
- Easy to integrate into existing websites.

❖ Applications:

- Registration forms for websites or apps.
- Feedback and survey forms.
- Online booking and e-commerce forms.

❖ Conclusion:

The **Interactive Form Validation** project successfully demonstrates how real-time validations improve usability and data integrity. By providing instant feedback, it ensures users submit **correct and complete information**, reducing errors and enhancing the overall experience.

❖ Future Scope:

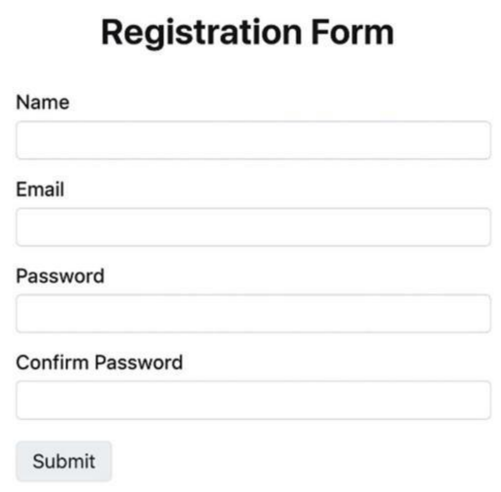
- Add **multi-language support**.
- Integrate **API-based server-side validation**.
- Add **AI-based suggestions** for inputs (e.g., email domain suggestions).
- Make it **mobile-friendly** with advanced UI effects.

❖ References:

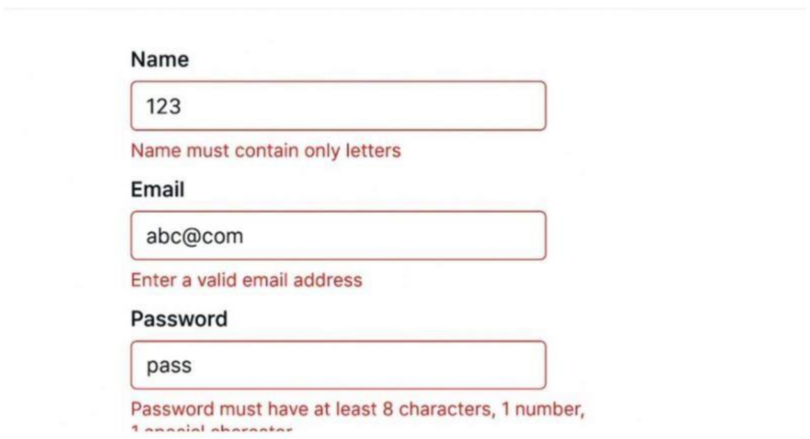
1. MDN Web Docs – HTML, CSS, JavaScript.
2. W3Schools – Form Validation Tutorial.

3. Screenshots /API Documentation:

❖ **Screenshot 1: Form Initial State:**

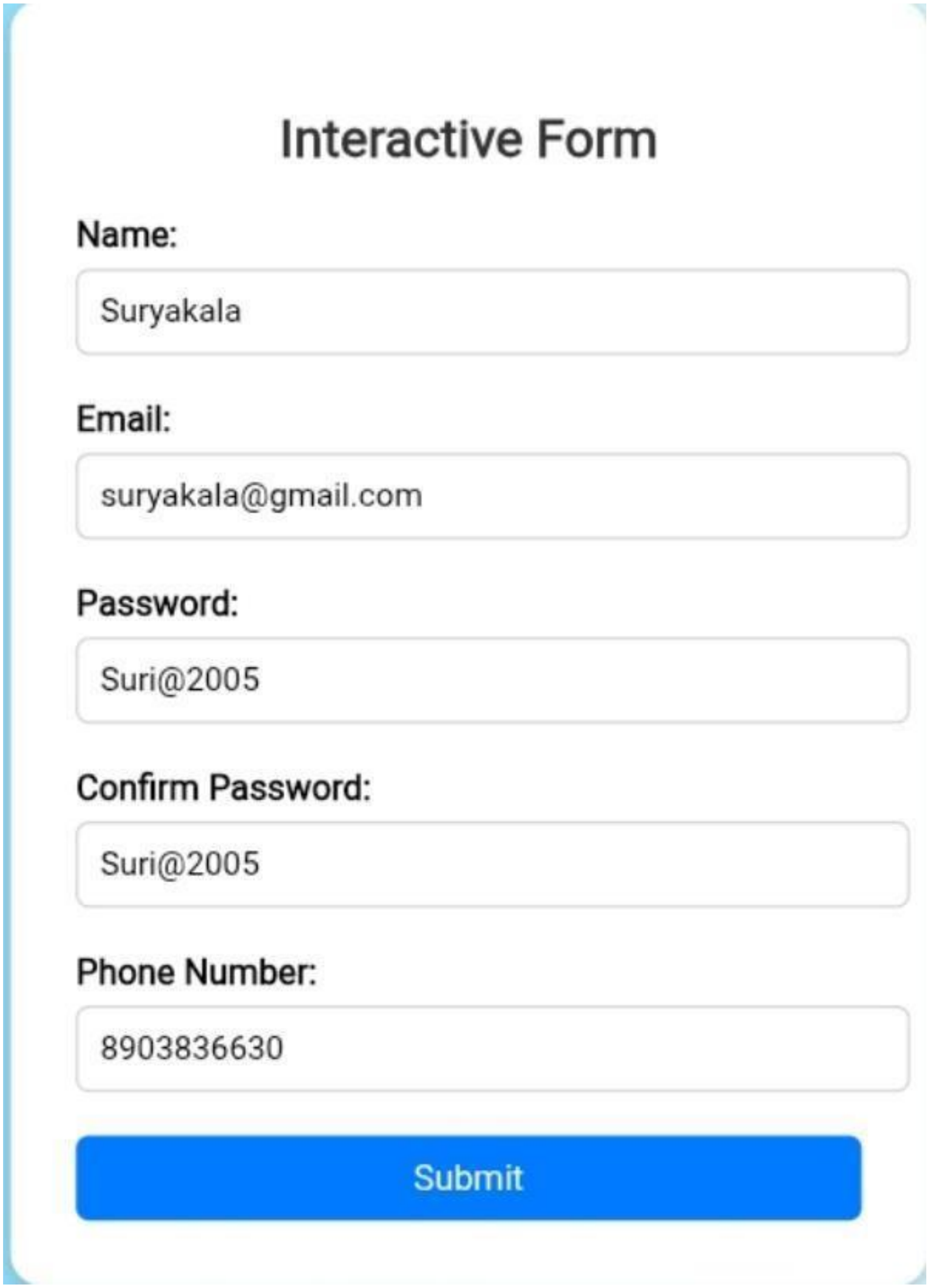


❖ Screenshot 2: Real-Time Validation – Errors:



123
Name must contain only letters
abc@com
Enter a valid email address
pass
Password must have at least 8 characters, 1 number,

❖ Screenshot 3: Real-Time Validation – Correct Inputs:



Interactive Form

Name:

Suryakala

Email:

suryakala@gmail.com

Password:

Suri@2005

Confirm Password:

Suri@2005

Phone Number:

8903836630

❖ Screenshot 4: Successful Form Submission:

Form submitted successfully!

❖ API / Function Documentation:

Function Name	Purpose	Input	Output / Effect
`validateName()`	Checks if the Name field contains only letters	`name.value` (string)	Shows error message if invalid, clears error if valid
`validateEmail()`	Checks if Email is in correct format	`email.value` (string)	Shows error message if invalid, clears error if valid
`validatePassword()`	Checks password strength (min 8 chars, number, special char)	`password.value` (string)	Shows which criteria are missing, clears message if valid
`validateConfirmPassword()`	Ensures Confirm Password matches Password	`password.value`, `confirmPassword.value`	Shows error if passwords do not match
`validatePhone()`	Checks if Phone Number has exactly 10 digits	`phone.value` (string)	Shows error if invalid, clears message if valid
`enableSubmit()`	Enables submit button if all fields are valid	N/A	Submit button becomes clickable
`handleSubmit(event)`	Handles form submission	Event object	Prevents submission if errors exist, shows success message if valid

4. Challenges and solutions:

❖ Challenge 1: Required Fields Validation:

Problem: Ensure that the user doesn’t leave the Name, Email, or Password fields empty.

Solution (Interactive):

```
<form id="form1">

  <label>Name:</label>

  <input type="text" id="name"><br>

  <label>Email:</label>

  <input type="email" id="email"><br>

  <label>Password:</label>
  <input type="password" id="password"><br>
  <button type="submit">Submit</button>
</form>
<p id="error1" style="color:red;"></p>

<script>
document.getElementById('form1').addEventListener('submit', function(e){
  e.preventDefault();
  let name = document.getElementById('name').value;
  let email = document.getElementById('email').value;
  let password = document.getElementById('password').value;
  let errorMsg = '';
  if(!name) errorMsg += 'Name is required. ';
  if(!email) errorMsg += 'Email is required. ';
  if(!password) errorMsg += 'Password is required. ';

  document.getElementById('error1').innerText = errorMsg || 'Form submitted successfully!';
});
</script>
```

❖ Challenge 2: Email Format Validation:

Problem: Ensure that the email entered is in a proper format.

Solution:

```
<form id="form2">
  <label>Email:</label>
  <input type="email" id="email2">
  <button type="submit">Submit</button>

</form>

<p id="error2" style="color:red;"></p>


<script>

document.getElementById('form2').addEventListener('submit', function(e){

  e.preventDefault();

  let email = document.getElementById('email2').value;

  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;


  if(!regex.test(email)){

    document.getElementById('error2').innerText = 'Please enter a valid email.';

  } else {

    document.getElementById('error2').innerText = 'Email is valid!';

  }

});

</script>
```

❖ Challenge 3: Password Strength Validation:

Problem: Password must be at least 8 characters, contain a number, an uppercase letter, and a special character.

Solution:

```
<form id="form3">

  <label>Password:</label>

  <input type="password" id="password3">
  <button type="submit">Submit</button>

</form>

<p id="error3" style="color:red;"></p>


<script>

document.getElementById('form3').addEventListener('submit', function(e){

  e.preventDefault();

  let password = document.getElementById('password3').value;

  let regex = /^(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$/;


  if(!regex.test(password)){

    document.getElementById('error3').innerText = 'Password must be 8+ chars, include 1 uppercase, 1
number & 1 special char.';

  } else {

    document.getElementById('error3').innerText = 'Strong password!';

  }

});

</script>
```

❖ Challenge 4: Confirm Password Match:

Problem: Ensure that “Confirm Password” matches the “Password” field.

Solution:

```
<form id="form4">

  <label>Password:</label>

  <input type="password" id="password4"><br>

  <label>Confirm Password:</label>
  <input type="password" id="confirm4"><br>

  <button type="submit">Submit</button>

</form>

<p id="error4" style="color:red;"></p>


<script>

document.getElementById('form4').addEventListener('submit', function(e){

  e.preventDefault();

  let password = document.getElementById('password4').value;

  let confirm = document.getElementById('confirm4').value;


  if(password !== confirm){

    document.getElementById('error4').innerText = 'Passwords do not match!';

  } else {

    document.getElementById('error4').innerText = 'Passwords match!';

  }

});

</script>
```

❖ Challenge 5: Phone Number Validation:

Problem: Phone number must be 10 digits and numeric.

Solution:

```
<form id="form5">

  <label>Phone Number:</label>

  <input type="text" id="phone5">

  <button type="submit">Submit</button>
```

```
</form>

<p id="error5" style="color:red;"></p>



<script>

document.getElementById('form5').addEventListener('submit', function(e){

  e.preventDefault();

  let phone = document.getElementById('phone5').value;

  let regex = /^\d{10}$/;



  if(!regex.test(phone)){

    document.getElementById('error5').innerText = 'Enter a valid 10-digit phone number.';

  } else {

    document.getElementById('error5').innerText = 'Phone number is valid!';

  }

});

</script>
```

5. GITHUB README AND SETUP GUIDE:

Project Overview:

In this project, I created an **interactive form** that checks user input **instantly while typing**. The main aim is to make form filling easier and error-free.

Whenever a user types something wrong — like an invalid email, weak password, or mismatched passwords — the form immediately shows an error message without reloading

Objective:

The goal of my project is to:

- Help users fill out forms correctly by showing live feedback
- Reduce mistakes before submitting the form

- Improve the overall user experience

Working Process (Step by Step):

1. Name Field:

- The form checks if the user has entered their name.
- If the field is left empty, it shows "Name is required."

This helps ensure no field is missed.

2. Email Field:

- The email entered is checked to make sure it has the correct format (example: user@gmail.com).
- If the email format is wrong, an error message appears instantly.

This makes sure users enter a valid email address.

3. Password Field:

- The password must follow certain rules:
  - At least 8 characters long
  - Must include one uppercase letter
  - Must include one number
  - Must include one special character (like @, #, $)
- The form immediately tells if the password is weak or doesn't meet the conditions.

This improves security by forcing users to choose strong passwords.

4. Confirm Password Field:

- This field checks if it matches the original password.
- If both passwords don't match, it shows "Passwords do not match."

This prevents login or registration problems later.

5. Phone Number Field:

- The form checks if the phone number has **exactly 10 digits** and contains only numbers.
- If not, it shows an error like "Enter valid 10-digit phone number."

This ensures that users enter a correct contact number.

6. Real-Time Validation:

- The main feature of my project is that all validations happen **live** while typing.
- Users don't need to click submit to see errors.
- Messages appear and disappear automatically as the user corrects input.

This makes the form more interactive and user-friendly.

7. Final Submission:

- When the user clicks the submit button:
    - The form checks all fields again.
    - If there are any errors, it shows a message like "Please fix the errors before submitting."
    - If everything is correct, it displays "Form submitted successfully!"

This confirms that all details are valid and ready to be submitted.

User Experience:

- Error messages are shown in red, and success messages in green.
- The layout is clean, modern, and easy to use.
- The design is responsive, so it works well on different devices.

What I Learned:

From this project, I learned:

- How to handle forms using HTML and JavaScript
- How to validate user input using regular expressions (regex)
- How to show messages dynamically using DOM manipulation
- The importance of creating user-friendly web forms

Where It Can Be Used:

This form validation system can be used in:

- Registration forms
- Login pages
- Feedback or contact forms
- Online booking or survey forms

Advantages:

Users get instant feedback
Reduces form submission errors
Saves time and improves accuracy
Makes the website look professional and interactive

Future Improvements:

In the future, I can add:

- Green/red icons next to each field
- A password strength meter
- Smooth animations for better UI
- Connection to a database to save form data
- Dark mode theme

6. FINAL SUBMISSION(REPO+DEPLOYED LINK):

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Interactive Form Validation</title>

<style>

body {

font-family: Arial, sans-serif;

background: linear-gradient(to right, #6dd5fa, #ffffff);

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

form {

background: #fff;
```

```
padding: 30px;

border-radius: 15px;

box-shadow: 0 5px 15px rgba(0,0,0,0.2);

width: 360px;

}

h2 {

text-align: center;

color: #333;

margin-bottom: 20px;

}

label {

font-weight: bold;

display: block;

margin-top: 10px;

}

input {

width: 100%;

padding: 10px;

margin: 6px 0 10px;

border: 1px solid #ccc;

border-radius: 6px;

font-size: 15px;

}

input:focus {

border-color: #007BFF;
```

```
outline: none;

}

.error {

color: red;

font-size: 13px;

height: 16px;

}

.success {

color: green;

font-size: 13px;

}

button {

width: 100%;

padding: 10px;

background: #007BFF;

color: white;

border: none;

border-radius: 6px;

font-size: 16px;

cursor: pointer;

margin-top: 10px;

}

button:hover {

background: #0056b3;

}
```

```
#submitMessage {

text-align:  center;

margin-top:  10px;

font-weight: bold;

}

</style>

</head>

<body>

<form id="liveForm">

<h2>Interactive Form</h2>


<label>Name:</label>

<input type="text" id="name" placeholder="Enter your name">

<div id="nameError" class="error"></div>


<label>Email:</label>

<input type="email" id="email" placeholder="Enter your email">

<div id="emailError" class="error"></div>


<label>Password:</label>

<input type="password" id="password"  placeholder="Enter password">

<div id="passwordError" class="error"></div>


<label>Confirm Password:</label>

<input type="password" id="confirm" placeholder="Re-enter password">
```

```
<div id="confirmError" class="error"></div>


<label>Phone Number:</label>

<input type="text" id="phone" placeholder="Enter 10-digit phone number">

<div id="phoneError" class="error"></div>


<button type="submit">Submit</button>

<div id="submitMessage"></div>

</form>


<script>

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

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

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

const confirmField = document.getElementById('confirm');

const phoneField = document.getElementById('phone');

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


// Name Validation

nameField.addEventListener('input', () => {

const name = nameField.value.trim();

document.getElementById('nameError').innerText =

name ? '' : 'Name is required.';

});
```

```
// Email Validation

emailField.addEventListener('input', () => {

const email = emailField.value.trim();

const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

document.getElementById('emailError').innerText =

!email ? 'Email is required.' : (regex.test(email) ? '' : 'Invalid email format.');

});


// Password Validation

passwordField.addEventListener('input', () => {

const password = passwordField.value;

const regex = /^(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$/;

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

if (!password) errorDiv.innerText = 'Password is required.';

else if (!regex.test(password))

errorDiv.innerText = 'Min 8 chars, 1 uppercase, 1 number & 1 special symbol.';

else errorDiv.innerText = '';

});


// Confirm Password Validation

confirmField.addEventListener('input', () => {

const confirm = confirmField.value;

const password = passwordField.value;

document.getElementById('confirmError').innerText =

confirm !== password ? 'Passwords do not match.' : '';
```

```
});

// Phone Number Validation
phoneField.addEventListener('input', () => {
const phone = phoneField.value;
const regex = /^\d{10}$/;
document.getElementById('phoneError').innerText =
!regex.test(phone) ? 'Enter valid 10-digit phone number.' : '';
});

// On Form Submit
form.addEventListener('submit', (e) => {
e.preventDefault();
const errors = document.querySelectorAll('.error');
const hasError = Array.from(errors).some(error => error.innerText !== '');
const emptyFields = [nameField, emailField, passwordField, confirmField, phoneField]
.some(field => !field.value.trim());

const message = document.getElementById('submitMessage');
if (hasError || emptyFields) {
message.style.color = 'red';
message.innerText = 'Please correct the errors before submitting.';
} else {
message.style.color = 'green';
message.innerText = 'Form submitted successfully!';
form.reset();

}

});

</script>

</body>

</html>
```

LINK:

https://github.com/dheeptha730/NM-phase5