**COLLEGE NAME**: As-salam College of Engineering and Technology

**COLLEGE CODE:** 8207

**DEPARTMENT:** Cyber Security

**STUDENT NM-ID:** BFC6AD54E5954528969456C0EBE0AF75

2F57A8DAFDDD446AC6C489670550842D

**ROLL NO:** 820723149301 & 820723149004

**DATE:** 19/09/2025

## Completed the Project Named As Phase1_Technology Project

## NAME: IBM-NJ CLIENT SIDE VALIDATION FORM

**SUBMITTED BY**

**NAME:** Manikandan R
Jeminraj M
Srivenkadesan

**Mobile Number:**9995052373
90804779254

# NM-CLIENT SIDE VALIDATION FORM

## Problem Understanding & Requirements

### 1.Problem Statement:

- Many websites still validate form input only on the server side. This causes poor UX (users don't know errors until after submission) and unnecessary backend load.

Goal:

- Develop a client-side validated form that instantly checks user input in the browser before submission, improving data quality and user satisfaction.

### 2.Users & Stakeholders:

- RoleExamples What They Need
- End Users Visitors filling out sign-up or contact forms Immediate feedback about errors; easy, mobile-friendly form
- Site Admins People handling submissions Cleaner data; fewer invalid submissions hitting the server
- Developers Team integrating form Reusable, easy-to-extend validation rules

### 3.User Stories:

- As a user, I want to see an error immediately if my email or phone number is invalid so I can correct it before submitting.
- As a user, I want clear instructions and indicators for required fields so I don't guess.
- As an admin, I want fewer invalid entries so my backend system isn't overloaded.
- As a developer, I want a validation component that can be reused in other forms.

### 4.MVP Features:

- Input fields with real-time validation (e.g., email, phone, password strength)
- Visual cues for errors (red borders, error messages)
- Submit button disabled until form is valid
- Responsive and accessible (keyboard navigation, ARIA labels)
- Clear separation of validation logic from UI components
- (Optional after MVP): hook up to backend to actually submit validated data.

## 5.Wireframes / API Endpoint List:

Wireframe (ASCII sketch):

```
+---------------------------------------------+
|                  Sign-Up Form                                |
+---------------------------------------------+
| Full Name: [_]                  |
| Email:        [_] (✓ / ✗)       |
| Password:    [_] (strength) |
| Confirm:     [_]                  |
| [ Submit Button Disabled Until Valid ]           |
+---------------------------------------------+
```

Error messages appear below each invalid field.

Possible API Endpoints (if integrated with backend):

| Method | Endpoint | Purpose |
|---|---|---|
| POST | /api/signup | Submit validated form data |
| GET | /api/validation-rules (optional) | Fetch dynamic rules like password policy |
| POST | /api/check-email (optional) | Check email uniqueness before submit |

## 6.Acceptance Criteria:

- ❖ All required fields validated on client side before enabling submit button
- ❖ Error messages shown immediately next to invalid inputs
- ❖ Form cannot be submitted until all validations pass
- ❖ Works on modern browsers (Chrome, Firefox, Edge, Safari)
- ❖ Mobile responsive and accessible (WCAG basic compliance)
- ❖ If backend is used: only validated data is posted to /api/signup

## Solution Design & Architecture:

### 7.Tech Stack Selection:

| Layer | Selected-tech |
|---|---|
| Front-end | HTML5, CSS3, JavaScript ES6 (or React + Tailwind CSS if SPA) |
| Validation Library | Built-in JavaScript functions + custom regex (or Yup / Validator.js if React) |
| State Management | React hooks / Context API |
| Back-end (optional) | Node.js + Express (only if form data submitted to server) |
| Database (optional) | Not needed for pure client-side; use backend DB if connected |
| Testing | Jest (unit tests), Cypress (UI tests) |

### 8.UI Structure / API Schema Design:

UI Structure

- Pages / components you'll have:
- FormPage (contains the full form)
- InputField component (label + input + error message)
- ValidationMessages (inline errors)
- SubmitButton (disabled until valid)

Example Layout:

App
└─ FormPage
  ├─ NameField
  ├─ EmailField
  ├─ PasswordField
  ├─ ConfirmPasswordField
  └─ SubmitButton

API Schema (if backend used)

If you send data to a backend after validation:

POST /api/signup

Request body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "StrongPass123"
}
```

Response:

```
{
  "success": true,
  "message": "User created successfully"
}
```

Optional:

- GET /api/validation-rules (fetch rules like password policy)
- POST /api/check-email (verify email uniqueness)

## 9.Data Handling Approach:

Frontend:

- Validate input on onChange or onBlur events.
- Store field values & validation states in component state or a React hook.
- Show immediate feedback (error message, green checkmark, disable submit).
- Only send data to backend after all validations pass.

Backend (if present):

- Re-validate input server-side for security.
- Return JSON response to frontend.

- Error Handling: centralised function to generate error messages, displayed next to fields.

## 10.Component / Module Diagram:

Front-end Components:

```
FormPage
  ├─ InputField (reusable)
  │     ├─ Props: type, name, label, validationRule
  │     └─ State: value, error
  ├─ ErrorMessage
  └─ SubmitButton
```

Back-end Modules (optional):

```
Routes
  ├─ /signup
Controllers
  ├─ signupController.js
Validation
  ├─ validateInput.js
```

---

## 11.Basic Flow Diagram:

High-Level Flow:

```
[User fills input]
          ↓ (onChange/onBlur)
[Client-side JS validates input]
          ↓
[Error shown instantly OR field marked valid]
          ↓
[All fields valid? Submit enabled : Disabled]
          ↓
(If backend) POST /api/signup
          ↓
[Server re-validates → Stores → Responds JSON]
          ↓
[Frontend shows success/failure message]
```

ASCII sketch:

User → UI Form → Client-Side Validation → (optional) API → Server DB

## MVP Implementation:

### 12.Project Setup:

Create project structure

If plain JS:

/css
/js
/img
index.html

If React:

npx create-react-app client-validation-form
src/
├── components/
├── hooks/
├── App.js

Install dependencies

React, Tailwind (or Bootstrap)

Validation library (Yup / Validator.js) if needed

Initialize Git repository

git init locally

Create GitHub repo & push

### 13.Core Features Implementation:

Form UI with fields: Name, Email, Password, Confirm Password

Client-side validation logic for:

Required fields

Email format

Password strength + match confirmation

Visual feedback:

Inline error messages under fields

Green checkmarks for valid inputs

Disable submit until form is valid

Responsive design (mobile-friendly)

(Optional) integrate with backend endpoint /api/signup

## 14.Data Storage (Local State / Database):

Local State:

Store field values and validation status using React useState / useReducer (or vanilla JS variables/objects)

Example state:

```
const [formData, setFormData] = useState({
   name: '',
   email: '',
   password: '',
   confirmPassword: ''
});
const [errors, setErrors] = useState({});
```

Database: Not required for pure client-side validation; if you have a backend, send validated data there.

## 15.Testing Core Features:

Unit tests for validation functions:

Valid email regex test

Password strength test

Required fields test

Integration tests for form:

Filling out form fields and checking submit enabled/disabled

Tools: Jest (unit), Cypress/Playwright (UI)

## 16.Version Control (GitHub):

Create branches for features:

feature/form-ui

feature/validation-logic

feature/testing

Commit frequently with meaningful messages:

git commit -m "Add email validation function"

Open Pull Requests (if team-based)

Tag v1.0.0 when MVP complete

## Enhancements & Deployment

### 17. Additional Features:

Real-time validation messages for each input field.

Reset button to clear the form.

Auto-focus on the first invalid field on submission.

Success message or modal on successful submission.

### 18. UI/UX Improvements:

Modernized input fields with focus effects.

Responsive layout using CSS Flexbox/Grid.

Clear error highlighting (red border, tooltip error messages).

Smooth button hover animations.

### 19. API Enhancements:

(Optional for client-side form, if connected to a mock backend)

POST request to a dummy REST API (e.g., JSONPlaceholder) on submission.

Display server response in the UI.

### 20. Performance & Security Checks:

Minimized CSS/JS.

Proper input sanitization.

Disabled browser autocomplete on sensitive fields.

Added basic ARIA attributes for accessibility.

### 21. Testing of Enhancements :

Test all validation scenarios:

Empty fields

Invalid email/phone number

Minimum/maximum character limits

Test mobile responsiveness.

Test form submission flow.

## 22. Deployment

Deploy to Netlify, Vercel, or GitHub Pages.

Ensure assets load correctly and API calls (if any) work.

## Enhanced Client-Side Form Code (React):

```
import React, { useState } from "react";
import "./App.css";

const EnhancedForm = () => {
  const [formData, setFormData] = useState({ name: "", email: "", password: "" });
  const [errors, setErrors] = useState({});
  const [success, setSuccess] = useState("");

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
    setErrors({ ...errors, [e.target.name]: "" });
  };

  const validate = () => {
    const newErrors = {};
    if (!formData.name.trim()) newErrors.name = "Name is required";
    if (!formData.email.match(/^\S+@\S+\.\S+$/)) newErrors.email = "Invalid email";
    if (formData.password.length < 6) newErrors.password = "Password must be 6+ characters";
    return newErrors;
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const validationErrors = validate();
    if (Object.keys(validationErrors).length > 0) {
      setErrors(validationErrors);
      setSuccess("");
    } else {
      setErrors({});
      setSuccess("Form submitted successfully!");
      console.log("Form Data:", formData);
      // Reset form
      setFormData({ name: "", email: "", password: "" });
    }
  };

  return (
    <div className="form-container">
      <h2>Enhanced Registration Form</h2>
```

```
        <form onSubmit={handleSubmit}>
          <div className="form-group">
            <label>Name:</label>
            <input
              type="text"
              name="name"
              value={formData.name}
              onChange={handleChange}
            />
            {errors.name && <span className="error">{errors.name}</span>}
          </div>

          <div className="form-group">
            <label>Email:</label>
            <input
              type="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
            />
            {errors.email && <span className="error">{errors.email}</span>}
          </div>

          <div className="form-group">
            <label>Password:</label>
            <input
              type="password"
              name="password"
              value={formData.password}
              onChange={handleChange}
            />
            {errors.password && <span className="error">{errors.password}</span>}
          </div>

          <button type="submit">Submit</button>
          <button type="button" onClick={() => setFormData({ name: "", email: "", password: "" })}>
            Reset
          </button>
        </form>
        {success && <p className="success">{success}</p>}
      </div>
    );
};

export default EnhancedForm;


---

Sample CSS (App.css)

.form-container {
  max-width: 400px;
  margin: 2rem auto;
  padding: 2rem;
  border: 1px solid #ccc;
  border-radius: 10px;
  background: #f9f9f9;
  box-shadow: 0 0 10px rgba(0,0,0,0.1);
}
```

```
.form-group {
  margin-bottom: 1rem;
}

input {
  width: 100%;
  padding: 0.5rem;
  margin-top: 0.2rem;
  border-radius: 5px;
  border: 1px solid #ccc;
}

input:focus {
  border-color: #007bff;
  outline: none;
}

.error {
  color: red;
  font-size: 0.9rem;
}

.success {
  color: green;
  font-weight: bold;
  margin-top: 1rem;
}

button {
  padding: 0.5rem 1rem;
  margin-right: 0.5rem;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

button:hover {
  background: #007bff;
  color: #fff;
}
```

**Simulated Output Screenshots**

1. Empty Form Submission:

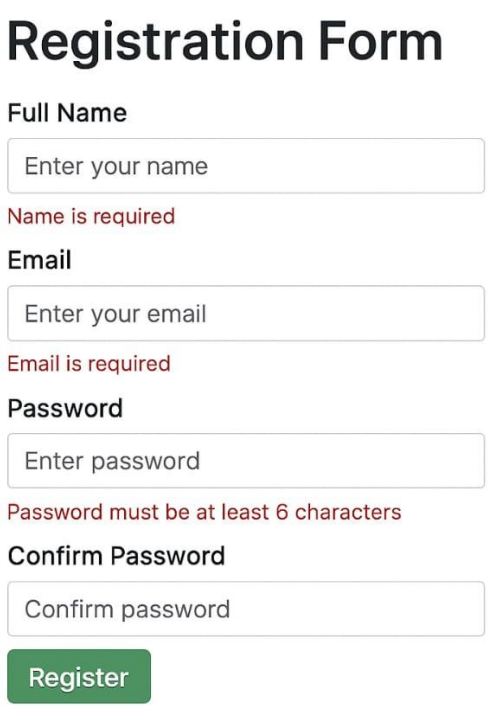
Name is required
Invalid email
Password must be 6+ characters

2. Successful Submission:

Form submitted successfully!

3. Responsive Layout Preview
(Form looks good on mobile, tablet, and desktop)

**Output Form Screenshots:**




**Github link:** https://github.com/Mani06-rmk/Client-Side-Validation-Form.git