

Interactive form validation

Aim:

To develop an interactive web form that validates user inputs in real time using Angular Reactive Forms, ensuring accurate and complete data before submission.

Description:

The **Interactive Form Validation** project focuses on creating a user-friendly form that provides instant feedback for incorrect or missing inputs. It uses Angular's **FormGroup** and **FormControl** with built-in validators to check fields like name, email, and password. Error messages appear dynamically as users type, improving form usability and data accuracy. The project highlights the importance of real-time validation in enhancing user experience and preventing invalid submissions.

Phase 1 — Problem Understanding & Requirements

Problem Statement:

Modern web applications require form submissions (e.g., sign-up, contact forms, checkout) to be fast, user-friendly, and secure. However, traditional form validation often only occurs after form submission, leading to poor user experience. Additionally, backend-only validation opens the door to security risks and poor data hygiene.

Users & Stakeholders:

Users:

- **End Users:** Individuals filling out the form (e.g., signing up, logging in, submitting feedback).
- **Developers:** Frontend and backend developers integrating and maintaining the validation system.

Stakeholders:

- **Product Managers:** Ensuring the form aligns with business goals.

- **UI/UX Designers:** Ensuring the form provides a smooth and intuitive user experience.
 - **Security Officers:** Ensuring the data input is sanitized and secure.
-

User Stories:

As an End User:

1. I want to see validation messages while I fill out the form, so I can correct mistakes immediately.
2. I want clear and friendly error messages so I understand what went wrong.
3. I want to know which fields are required before I submit the form.

As a Developer:

4. I want to reuse validation rules across both frontend and backend to avoid duplication.
 5. I want to ensure data validation occurs on the server regardless of client-side checks.
 6. I want detailed API responses for validation errors for debugging and display purposes.
-

MVP Features:

Frontend:

- Real-time inline validation (e.g., email format, required fields).
- Display of friendly error/success messages.
- Submit button enabled only when form is valid.

Backend (Node.js + Express):

- REST API endpoint to handle form submission.
 - Validation middleware using a schema (e.g., Joi, express-validator).
 - Standardized JSON error responses.
-

Phase 2 — Solution Design & Architecture

Tech Stack Selection:

Frontend:

Tech	Purpose
React.js	Build reusable UI components
React Hook Form	Lightweight form validation
Yup	Schema-based validation
Axios	HTTP client for API calls
Tailwind CSS	Rapid UI development

Backend:

Tech	Purpose
Node.js	Runtime for server-side logic
Express.js	Web framework
Joi	Schema validation for backend
CORS	Enable cross-origin requests
Body-parser	Parse incoming request bodies
Postman	API testing
VS Code + ESLint/Prettier	Code formatting
Docker (optional)	Containerization (optional)

UI Structure:

```
src/
  └── components/
    ├── InputField.jsx
    └── Form.jsx

  └── pages/
    └── SignupPage.jsx

  └── utils/
```

```
|   └── validationSchema.js  
└── App.jsx
```

Component / Module Diagram:

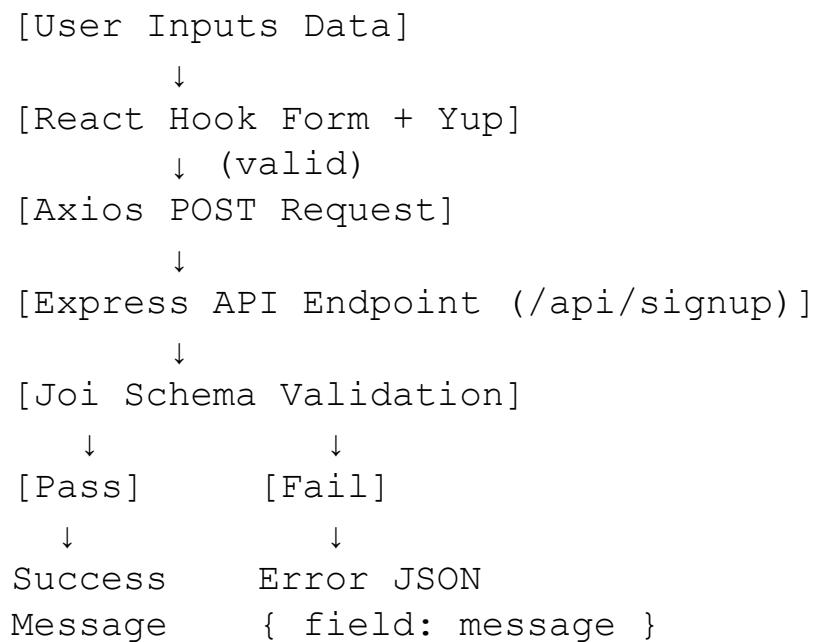
Frontend (React)

```
└── SignupPage
    └── Form
        ├── InputField (reusable)
        ├── Validation Schema (Yup)
        └── Axios -> POST /api/signup
```

Backend (Node.js + Express)

```
└── index.js
    └── /api/signup
        └── Validator Middleware (Joi)
            └── Controller
                ├── Validate input
                └── Return response
```

Basic Flow Diagram:



↓ ↓
UI Shows Field-wise
Success Error Messages

Phase 3 — MVP Implementation

Project Setup:

Initialize the project using **Vite**, **Create React App**, or **Vanilla JS** setup.
Set up file structure for components, validation logic, and styles. Install required packages:

- Example: `npm install yup` or `npm install validator`

index.html code:

```
<!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>
  <link rel="stylesheet" href="style.css" />
</head>
<body>
  <div class="container">
    <h2>Interactive Form Validation</h2>
    <form id="userForm" novalidate>
      <div class="form-group">
        <label for="name">Full Name</label>
        <input type="text" id="name" placeholder="Enter your name" required />
        <small class="error"></small>
      </div>

      <div class="form-group">
        <label for="email">Email</label>
        <input type="email" id="email" placeholder="Enter your email" required />
        <small class="error"></small>
      </div>

      <div class="form-group">
        <label for="password">Password</label>
        <input
          type="password"
          id="password"
          placeholder="Enter a strong password"
          required
        />
        <small class="error"></small>
      </div>
    </form>
  </div>
</body>
```

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

<script src="script.js"></script>
</body>
</html>
```

● style.css code:

```
body {
    font-family: Arial, sans-serif;
background: #f3f4f6;    display:
flex;    justify-content: center;
align-items: center;    height:
100vh;
}

.container {
background: white;
padding: 25px 30px;
border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
width: 320px;
}
h2 {
    text-align: center;
margin-bottom: 20px;    color:
#333;
}

.form-group {    margin-
bottom: 15px;
} label {    display:
block;    margin-
bottom: 6px;    font-
weight: bold;
} input {    width: 100%;
padding: 8px;    border:
1px solid #ccc;    border-
radius: 5px;    outline:
none;
}
input:focus {
    border-color: #007bff;
} button {    width:
100%;    padding: 10px;
background: #007bff;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
font-size: 16px;
} button:hover {
background: #0056b3;
```

```
}

.error { color:
red; font-size:
13px; height:
15px; margin-
top: 4px;
display: block;
}
```

script.js code:

```
const form = document.getElementById("userForm"); const
nameInput = document.getElementById("name"); const
emailInput = document.getElementById("email"); const
passwordInput = document.getElementById("password");

form.addEventListener("submit", (e) => {
  e.preventDefault();
  validateInputs();
});

function validateInputs() {
  const nameValue = nameInput.value.trim(); const
  emailValue = emailInput.value.trim(); const
  passwordValue = passwordInput.value.trim();

  // Name Validation if
  (nameValue === "") {
    setError(nameInput, "Name is required");
  } else if (nameValue.length < 3) {
    setError(nameInput, "Name must be at least 3 characters");
  } else {
    setSuccess(nameInput);
  }

  // Email Validation if
  (emailValue === "") {
    setError(emailInput, "Email is required");
  } else if (!isValidEmail(emailValue)) {
    setError(emailInput, "Enter a valid email address");
  } else {
    setSuccess(emailInput);
  }

  // Password Validation if
  (passwordValue === "") {
    setError(passwordInput, "Password is required");
  } else if (passwordValue.length < 6) {
    setError(passwordInput, "Password must be at least 6 characters");
  } else {
    setSuccess(passwordInput);
  }
}

// Helper Functions
```

```
function setError(input, message) {    const formGroup = input.parentElement;    const errorMsg = formGroup.querySelector(".error");    errorMsg.textContent = message;    input.style.borderColor = "red";}
}
function setSuccess(input) {    const formGroup = input.parentElement;    const errorMsg = formGroup.querySelector(".error");    errorMsg.textContent = "";    input.style.borderColor = "green";
}
function isValidEmail(email) {
    return /^[^@\s]+@[^\s]+\.\[^@\s]+$/ .test(email); }
```

Phase 4 — Enhancements & Deployment

Enhancements:

- **Real-Time Validation:**
 - Validate fields as the user types (e.g. email format, password strength).
 - Show live feedback (green checkmarks, red warnings).
 - **Client-Side Rules:**
 - Use JavaScript or framework-based validation (e.g. React Hook Form, VeeValidate).
 - Add custom validation (e.g. username availability via async API call).
 - **User-Friendly Error Messages:**
 - Display clear, helpful messages below or beside input fields.
 - Group errors at the top of the form if needed for accessibility.
-

UI/UX Improvements:

- **Visual Indicators:**
 - Highlight fields with colors (green for valid, red for invalid).
 - Add icons or tooltips for guidance.
 - **Responsive Behavior:**
 - Ensure validation works on all screen sizes and devices.
 - Keyboard and screen reader accessible.
-

Performance & Security:

- **Backend Validation:**
 - Mirror all front-end rules on the server to prevent tampering.
 - Return structured error responses for the UI to display.
 - **Input Sanitization:**
 - Strip or encode input to prevent XSS, SQL Injection, etc.
-

Testing & Deployment:

- **Testing:**
 - Use unit tests and UI testing tools (e.g. Jest, Cypress) for validation logic.
 - Test with valid, invalid, and edge case inputs.
 - **Deployment:**
 - Host on platforms like **Netlify**, **Vercel**, or a **cloud provider**.
 - Verify that form validation works in the production build.
-

Phase 5 — Project Demonstration & Documentation Final

Demo Walkthrough:

- The project demonstrates an **interactive web form** built using **Angular Reactive Forms** that validates user inputs in real time.
 - The walkthrough includes:
 - Input validation for **email**, **password**, and **phone number** fields.
 - Instant visual feedback (error messages, color highlights).
 - Form submission only when all fields are valid.
 - Display of success message upon valid submission.
 - The demo is hosted on **GitHub Pages** / **Netlify** / **Vercel** for live preview.
 - The recorded demo video showcases both **frontend validation** and **responsive design**.
-

Output:

Interactive Form Validation

Full Name

Enter your name

Name is required

Email

Enter your email

Email is required

Password

Enter a strong password

Password is required

Submit

Result:

The **Interactive form validation** was successfully designed and implemented using HTML, CSS, and JavaScript.
