Interactive Form Validation

Phase 1 — Problem Understanding & Requirements

By
Aruna M - 2023103079
Harsika V - 2023103588
Sharan Saminathan - 2023103609
Sri Bavan Akash S - 2023103627
Paril T - 2023103714

Problem Statement

Many web forms still rely on server-side validation or static client-side checks that provide poor feedback, causing user frustration, higher abandonment rates, and incorrect data.

The goal of this project is to build an Interactive Form Validation frontend that:

- Provides fast, accessible, and user-friendly validation feedback in real-time
- Reduces form submission errors
- Improves conversion and completion rates

Scope: A single-page frontend application demonstrating robust validation for common form types (registration, login, contact, profile update), with extensible validation rules and excellent UX and accessibility.

Users & Stakeholders

Primary Users

- End users filling forms (students, job applicants, customers)
- Novice users who need clear guidance and accessible UI

Secondary Users / Stakeholders

- Product owner / course instructor (approves deliverables)
- UX designer (reviews wireframes & accessibility)
- Frontend developers (implement & maintain)
- QA/testers (validate behavior & edge cases)
- Backend developer (may provide API endpoints for submission/real-time checks)

User Stories

1. As a user, I want real-time inline validation so I can correct mistakes while typing.
2. As a user, I want helpful error messages that explain how to fix my input.
3. As a user, I want form fields to show clear success states once valid.
4. As a user with assistive tech, I want accessible validation (ARIA, announcements) so screen readers can understand errors.
5. As a product owner, I want the validation logic to be configurable so the same component can be reused across forms.
6. As a tester, I want a test suite of input scenarios so I can verify edge cases and regressions.
7. As a developer, I want the project to be modular and documented for easy maintenance.

MVP Features

Real-time validation (on input and on blur) for fields:

- Email (format + MX-like structure)
- Password (strength rules: min length, uppercase, lowercase, number, special char)
- Confirm password (matches)
- Name (non-empty, max length)
- Phone number (pattern validation, optional country code)
- Date (valid date and range)
- Custom text (e.g., username: allowed characters and uniqueness check via mock API)

Additional Features:

- Inline error messages and success indicators
- Accessible error linking (aria-describedby, role="alert" where appropriate)
- Prevent submission until all required fields are valid
- Form-level summary of errors on attempted submit
- Modular validation utility library (functions / rules)
- Unit tests for validation rules

Wireframes / API Endpoint List

Wireframes (Low-Fidelity Descriptions)

Registration Form — Layout (Single Column)

- Header: "Create an account"
- Inputs (stacked): Full name, Email, Username, Password, Confirm password, Phone number, Date of birth, Submit button
- Each input: label, field, inline helper / success / error text
- Below submit: form-level error summary container (collapsible)

Contact Form — Layout (Two Columns on Wide Screens)

- Left column: Name, Email, Subject
- Right column: Message textarea, Attachment (optional)
- Bottom: Submit button and confirmation toast

API Endpoint List (Mock / Optional for MVP)

- POST /api/validate-username — payload { username: string } → returns { available: boolean } (used for uniqueness check; can be mocked)
- POST /api/submit-form — payload: full form data → returns { success: boolean, errors?: {...} }

Acceptance Criteria

- All required fields show validation feedback in real-time and on blur.
- Error messages are human-readable and suggest corrective action (e.g., "Use at least 8 characters, including one number").
- Form cannot be submitted while validation errors exist.
- Screen reader users receive error announcements; the errors are navigable via aria-describedby.
- Visual success and error states meet contrast and design accessibility guidelines.
- Validation logic is covered by unit tests (≥ 80% for critical validation functions).
- The UI is responsive (mobile-first) and works on modern browsers.