COLLEGE CODE :9607

COLLEGE NAME :Immanuel Arasar JJ College Of Engineering

DEPARTMENT :B.TECH IT

STUDENT NM-ID :aut960723205003

ROLL NO :960723205003

DATE :23/09/2025

Complted the project named as phase 3

S.Kiruthika , S.Siva priya , M.Mathan , R.Palanisamy , D.Mugilan

Phase_TECHNOLOGY PROJECT NAME:

INTERACTIVE FORM VALIDATION

Submitted by,

Name : S.Kiruthiga

Mobile no : 6380834476

INTERACTIVE FORM VALIDATION

Phase 3

MVP Implementation

Contents

- Project setup
- Core features implementation
- Data storage(Local/Database)
- Testing core features
- Version control(GitHub)

Introduction:

- A minimum viable product(MVP)In today's digital experiences, from sign-ups and contact pages to surveys and checkout.
- This MVC focuses on building a simple , functional form with real-time validation-providing users immediate feedback as they fill out each field.

❖ Project Setup:

1.Choose Your Platform

Decide where you want to enable or share your form

- On your website?
- Shared via a link?
- Embedded in an email?

2.Create the Form

Go to the form builder and start a new form.

Add common fields:

- Full name
- Email
- Phone number
- Message
- Dropdown or multiple choice

3.Set up validation rules

- Required fields : Mark name and email as required.
- Email format check:Enable email validation
- Character limits:Set max/min length if needed.
- Conditional logic:Show hide fields based on answers.

4.Customize design & message:

- Adjust fonts /colors to match your brand.
- Add friendly error messages
- Set up thank you page or redirect after submission.

5.Test the form:

- Leaving requiredfields empty.
- Entering an invalid email.
- Submitting without filling optional fields.

6.Share or Embed:

- Use the share link.
- Or copy the embed code to ad it to your website.

7.Collect& View Response:

- View in a dashboard.
- Export to google sheets or Excel.
- Set up notification when someone submits.

❖ Core Features implementation:

- Instantaneous Feedback:

  When users enter data in a form field (such as a password or email address), the form instantly verifies the information.

  A notice is displayed immediately in the event of a problem (such as an incorrect email format), enabling them to address it immediately.

2. Validation of Individual Fields:

  Every input field—such as name, email, phone number, etc.—is examined separately.

  When the user begins typing, completes the text, or advances to the next field, validation may take place.

. 3 .Unambiguous Visual Signals:

  It is possible for fields to change color (red for errors, green for accurate).

  Users are guided by brief helper messages or tiny indicators .

4. Messaging Errors:

A brief warning (such as "Password must be at least 8 characters") indicates the reason if something is incorrect.

Good validation explains what is wrong and how to solve it, not just that it's "wrong."

5. Turned off the submit button till it was ready:

Occasionally, the submit button remains inactive until all needed fields have been filled out.

. 6. Support for Accessibility:

Additionally, the validation system ought to be compatible with keyboard navigation and screen readers.

Guarantees that any user, irrespective of aptitude, can comprehend the issue.

7. Design That Responds:

Validation functions flawlessly across all platforms, including desktop, tablet, and mobile.

❖ Data Storage (Localstate/Database):

Local State for Form Validation

Local state (e.g., in React) is commonly used for real-time, client-side validation as users fill out a form.

Use cases:

- Checking required fields

- Validating format (e.g., email, phone number)
- Comparing fields (e.g., password and confirm password)
- Enabling/disabling the submit button based on form state

Benefits:

- Fast feedback (no network latency)
- Better user experience (errors show immediately)
- Reduces unnecessary server calls

Database for Form Validation

Database-backed validation often occurs during form filling via API calls or upon submission. It is for rules that are not subject to local validation.

Examples of use:

- Verifying whether an email address or username is already taken
- Verifying coupons or discounts
- Cross-checking information (such as whether a user is present or whether a product is stocked)

Benefits:

- Secure and accurate (truth source)
- Stops people from using client-side modification to get around validation.

Combination Use :

- Both are used in the majority of robust forms:
- Local state for quick and easy verification.
- Validation of servers and databases for data integrity and business logic.

For instance:

- You verify the format locally as the user types an email.
- You make a request to the server to see if it's already registered when they stop typing or blur the field.

❖ Testing Core Features

1 .Client-side (local) validation

- When required fields are left empty, errors occur.
- Correct validation is performed on field forms (e.g., email regex, password length).
- When fields get input from the user, they respond appropriately (errors shown/cleared).
- Dependent fields validate against one another, such as "confirm password."

2.Server-Side/Database Validation:

- Correct validation requests, such as uniqueness checks, are sent by the form.

- Both legitimate and invalid responses—such as "username already taken"—are handled appropriately by the system.

- When performing async operations (such as loading indications or disabling submit), the form functions as intended.

- The user gets shown the relevant error message when the server returns an error.

3.Test Types to Employ:

- Unit Tests
- Test the isValidEmail and other validation routines.
- Examine local state management, such as component state or form state reduction.

4.Tests of Integration:

- Act as though you are completing a form.
- For async validation, use simulated API replies.

❖ version control (GitHub)

1 .Branches

❖ Create a distinct branch specifically for the form validation function.

keeps your work separate and unaffected by the production or main branches.

simplifies the process of testing and reviewing modifications prior to

merging.

2. Making a commitment

- ❖ Make brief, precise, and targeted commitments outlining the changes you made.
- ❖ Examples include "improving user feedback on invalid inputs," "fixing error messages," and adding email validation.
- ❖ keeps history comprehensible and facilitates debugging.

3. PRs or pull requests

- To merge your branch into the main or development branch after the feature is complete, open a PR.
- Use PRs to review code, debate changes, and obtain input from coworkers .
- Give concise explanations and background information regarding the purpose of your form validation and your testing methodology.

4. CI or continuous integration

- Every time you publish code or file a PR, use automated testing tools that are integrated with GitHub to execute your tests.
- makes sure your form validation logic is sound and doesn't interfere with other app features.

5. Versioning and Releases

- Create a release to designate a stable version of your application that incorporates the form validation feature after merging and testing.
- This makes it easier to monitor changes over time and reverse them if needed.

GitHub:

https://github.com/kiru2013/Kiruthiga