COLLEGE CODE : 9222

COLLEGE NAME : THENI KAMMAVAR SANGAM COLLEGEOFTECHNOLOGY

DEPARTMENT : B.tech(IT)

STUDENT NM-ID : ADC7870C46CFF6DA10D421BCC4347BFB

ROLL NO : 23it049

DATE : 08-10-2025

Completed the project named as

Phase__3 TECHNOLOGY PROJECT

NAME : **Client side form validation**

SUBMITTED BY,

NAME : M.Sundaramahalingam

MOBILE NO : 9360128907

MVP Implementation Plan for Client-Side Form Validation :

1. Project Setup

- Initialize a new project folder.
- Setup the development environment:
  - Choose your framework/library (React, Vue, Angular, or plain JS).
  - Setup package.json with necessary dependencies (e.g., React + Formik/Yup or vanilla JS).
- Initialize Git repository and create a GitHub repo.

2. Core Features Implementation

- Design a form with essential fields (e.g., Name, Email, Password, Confirm Password).
- Implement **client-side validation rules**:
  - Required fields.
  - Email format validation.
  - Password strength & matching confirm password.
  - Real-time validation feedback (error messages, success indicators).
- Use libraries if preferred (e.g., Formik with Yup for React).

3. Data Storage (Local State / Database)

- Store form data in local component state or a state management system (Redux, Vuex).
- For MVP, no backend is required; focus on local state.
- Optionally, simulate data persistence in localStorage to keep data between reloads.

4. Testing Core Features

- Manual testing of validation rules:
  - Submit invalid data, check error messages.
  - Submit valid data, ensure form submits correctly.
- Write basic unit tests if possible:
  - Validate individual validation functions.
  - Use Jest + React Testing Library or equivalent for other frameworks.
- Ensure edge cases are covered.

5. Version Control (GitHub)

- Regularly commit changes with clear messages.
- Use branches for major features or fixes.
- Push commits to GitHub.
- Create a README documenting:
  - Project setup instructions.
  - How to run and test the app.
  - Overview of validation rules implemented.

Testing Strategies for Client-Side Form Validation

1. Unit Testing Validation Logic

Test the pure validation function separately to ensure it catches errors correctly.

2. Component Testing

Test the form component renders correctly, displays validation messages, and reacts to user input.

3. Integration Testing

Test user flows such as filling out the form, submitting, and seeing success or error states.

Example:

Testing the Validation Function

```
export const validate = (form) => {
  const errors = {};

  if (!form.name.trim()) {
    errors.name = "Name is required";
  }
  if (!form.email) {
    errors.email = "Email is required";
  } else if (!/\S+@\S+\.\S+/.test(form.email)) {
    errors.email = "Email address is invalid";
  }
  if (!form.password) {
    errors.password = "Password is required";
  } else if (form.password.length < 6) {
    errors.password = "Password must be at least 6 characters";
  }
  if (!form.confirmPassword) {
    errors.confirmPassword = "Please confirm your password";
  } else if (form.password !== form.confirmPassword) {
    errors.confirmPassword = "Passwords do not match";
  }
  return errors;
```

};

Summary of Testing Types :

Type	What to test	Tools	Example
Unit Testing	Validation logic function only	Jest	Test `validate()` returns expected errors
Component Testing	Render and interaction of form	React Testing Library + Jest	Check inputs, error messages, success message
Integration Testing	User flow with multiple components	Cypress, Playwright (optional)	Simulate filling form and submitting in real app