

COLLEGE CODE : 9623

COLLEGE NAME : Amrita College of Engineering and
Technology

DEPARTMENT : Computer Science and Engineering

STUDENT NM-ID : 752E80CE0530AB3B286425E0EC83A2E0

ROLL NO : 962323104018

DATE :12-10-2025

COMPLETED THE PROJECT NAME AS PHASE 5

TECHNOLOGY PROJECT NAME: INTERATIVE FORM
VALIDATION

SUBMITTED BY,

NAME: ANUSH KUMARI M.S.

MOBILE NO: 9790054171

PROJECT PHASE 5: INTERATIVE FORM VALIDATION

Project Overview

The project titled “Interactive Form Validation” aims to develop a responsive and intelligent web-based form validation system that provides real-time feedback to users, ensures secure data submission, and enhances user experience. Forms are the most commonly used interface elements on websites and applications, but without proper validation, they often lead to incorrect data collection, security threats, and poor usability. This project focuses on building a validation mechanism that operates on both the frontend and backend, making sure that user inputs are correct, complete, and safely processed before being stored in the database. The validation system is implemented using modern technologies such as React.js, Node.js, and MongoDB, with additional libraries like React Hook Form, Yup, Express, Joi, and bcrypt for data handling, schema validation, and password encryption.

The project was divided into multiple phases to ensure systematic development and improvement.

PHASE 1 – PROBLEM UNDERSTANDING AND REQUIREMENTS

In the first phase, the main goal was to analyze the problem, identify the users, and define the key requirements of the system. It was observed that invalid form submissions can cause security issues and affect user trust. The system was designed to fix these challenges by ensuring data accuracy and user-friendly validation messages.

Objectives and Key Features:

- Real-time validation for required fields such as name, email, password, and phone number.

□

□

- Inline error messages that clearly explain the issue (e.g., “Email format invalid”, “Password must include 8+ characters”). REST API endpoints for validation and data submission.
- Database integration for storing validated submissions.
- Responsive and accessible user interface that works across all devices.

API Endpoints Designed:

- POST /form/validate – Validates form data.
- POST /form/submit – Submits and stores validated data.
- GET /form/submissions – Fetches submitted data for admin users.

Acceptance Criteria:

- Frontend real-time field validation.
- Server-side revalidation.
- Clear and user-friendly success/error messages.
- Secure data handling and storage.
- Responsive layout following designed wireframes.

Phase 2 – Solution Design & Architecture

- Tech Stack Used:
- Frontend – React.js, Tailwind CSS, React Hook Form / Formik
- Backend – Node.js with Express.js
- Database – MongoDB or PostgreSQL
- Validation – Yup / Joi
- Authentication – JWT and bcrypt (optional)
-
- UI / API Design:
- The form includes validation feedback, error tooltips, and submit button.
- API handles user data, re-validates input, and stores information in the database.
-
- Data Handling:
-
-

- Frontend validation is performed using React Hook Form and Yup.
- Backend validation is handled using Joi for additional security.
- Passwords are hashed before saving to the database.

Modules:

- Frontend – Validation Module, Error Handling Module, Submission Module
- Backend – Validation Middleware, Submission Controller, Admin Controller

□

- Flow of Operation:
- User enters data → Frontend validates input → Request sent to backend →
- Backend re-validates → Data stored in database → Success or error message returned

PHASE 3 – MVP IMPLEMENTATION

In this phase, the actual coding and implementation of the core modules were completed. The project was set up with proper version control using GitHub, including main and development branches to track all changes effectively.

Setup and Tools Used:

- Frontend: React.js with Tailwind CSS for responsive UI.
- Backend: Node.js with Express.js.
- Database: MongoDB for storing validated user submissions.
- Validation Libraries: React Hook Form, Yup, and Joi for schema validation.
- Password Security: bcrypt for password hashing before saving to the database.

Core Functionalities Implemented:

1. Real-time validation during form filling.
2. Display of inline error messages for incorrect inputs.
3. Secure API endpoints to handle data submission and validation.
4. Matching validation logic between frontend and backend.
5. Proper feedback provided for both successful and failed submissions.

Testing and Validation:

- Frontend tested using invalid and edge case inputs.
- Backend tested using Postman to verify response codes and data storage.

□

□

Database tested to confirm only valid submissions are saved.
Version control maintained using GitHub commits at each milestone.

-
-

Deliverable of Phase 3:

A fully functional MVP (Minimum Viable Product) capable of interactive form validation, real-time feedback, and secure data submission.

PHASE 4 – ENHANCEMENTS AND DEPLOYMENT

This phase focused on improving the existing system with advanced features, performance optimization, and final deployment on a cloud platform such as Netlify or Vercel.

Enhancements Added:

- Comprehensive validation across all form modules (registration, login, feedback).
- Real-time validation using JavaScript to instantly alert users about incorrect input.
- Custom, descriptive error messages for better guidance.
- Auto-focus and reset features for invalid fields to improve usability.
- Prevention of form submission until all fields are valid.

UI/UX Improvements:

- Refined form layout with icons, labels, and placeholders.
- Highlighted invalid fields with red borders and tooltips.
- Displayed green checkmarks (✓) for valid fields.
- Mobile-responsive validation interface.
- Inline messages instead of intrusive pop-ups for smooth user experience.

API and Security Enhancements:

- Backend validation reinforced with sanitization and encoding to prevent SQL injection and XSS attacks.
- Middleware added for error logging and response tracking.
- Field dependency checks implemented (password confirmation, etc.). □ Rate limiting and CAPTCHA integrated to prevent spam submissions.
- Password encryption and hashing applied following OWASP standards.

Testing and Deployment:

- Conducted unit and integration testing using Jest and Postman.
- Load testing performed to ensure scripts did not affect performance.
- User Acceptance Testing (UAT) done with QA team.

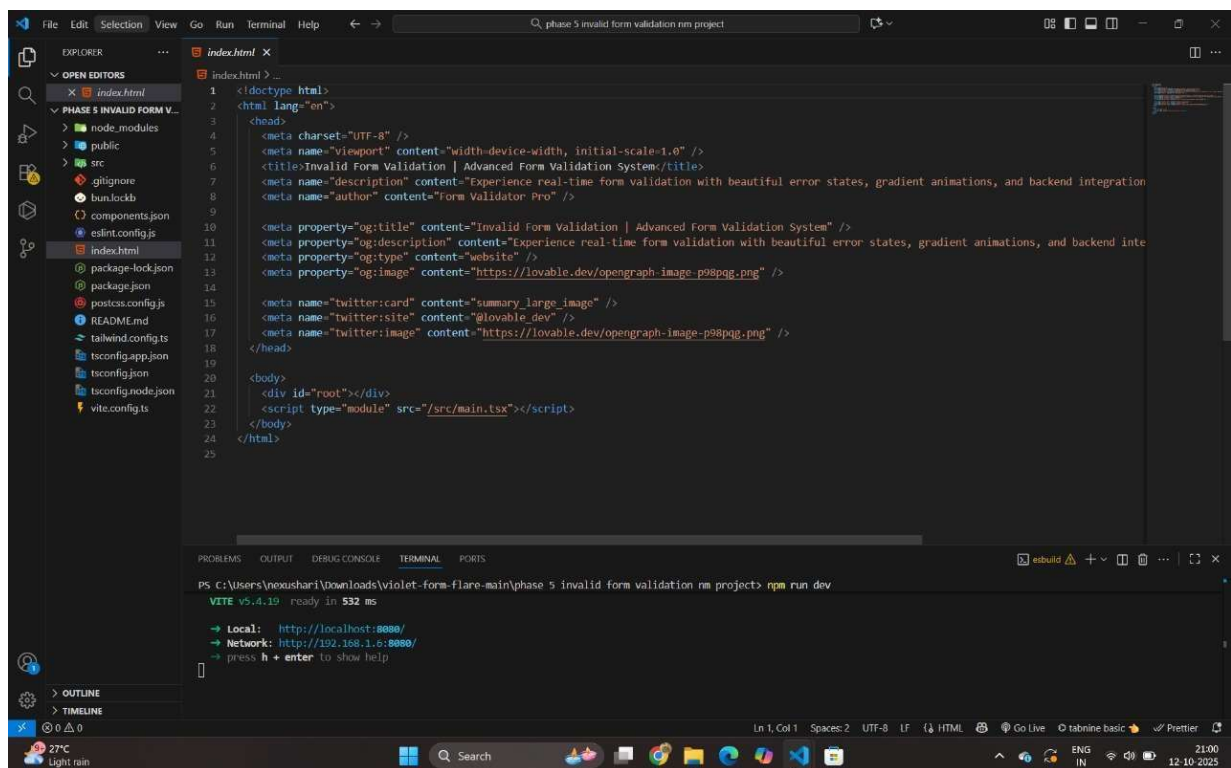
□

- Successfully deployed and tested across multiple browsers and devices.

Final Outcome:

A secure, stable, and responsive Interactive Form Validation System successfully deployed and verified for public use.

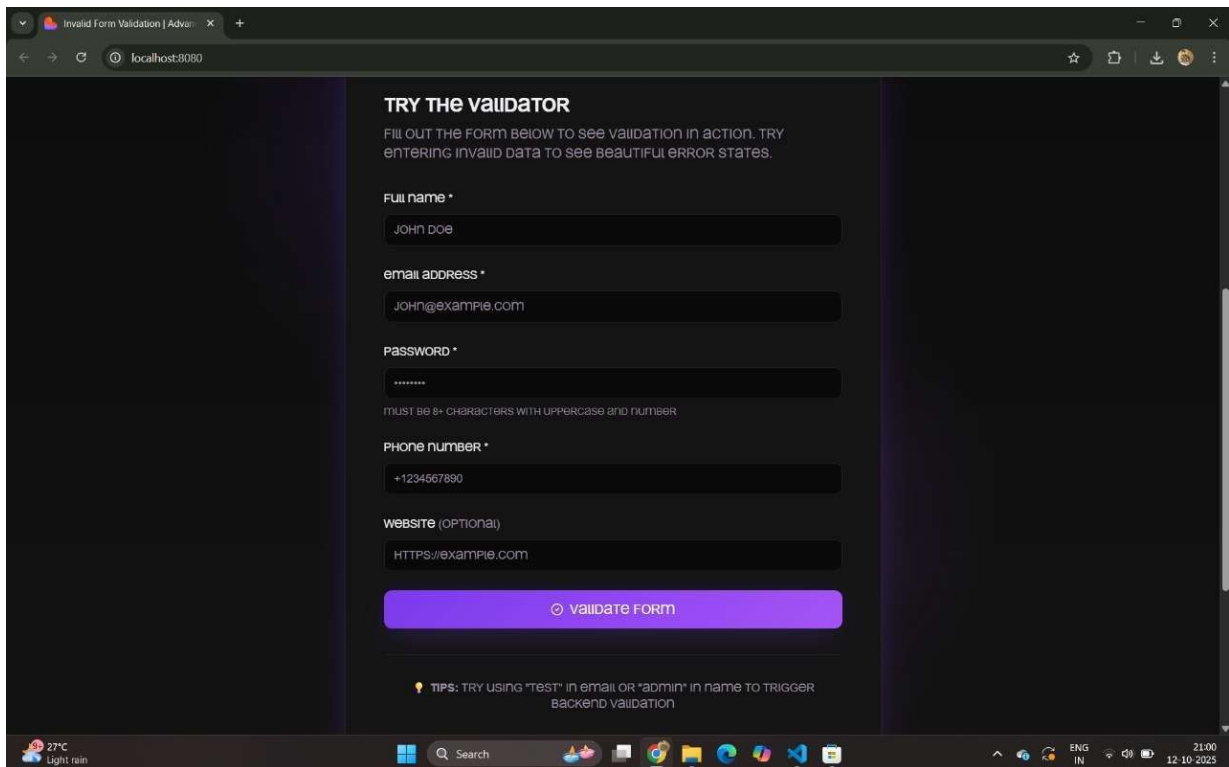
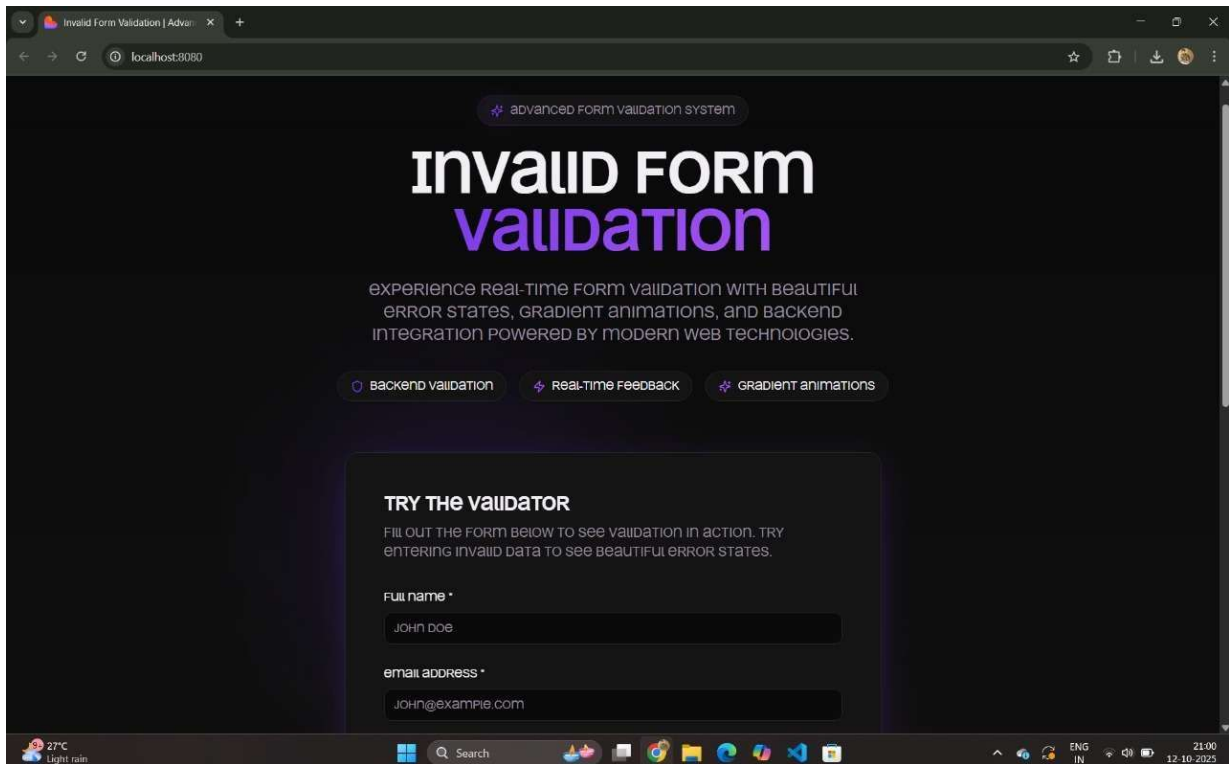
Screenshots

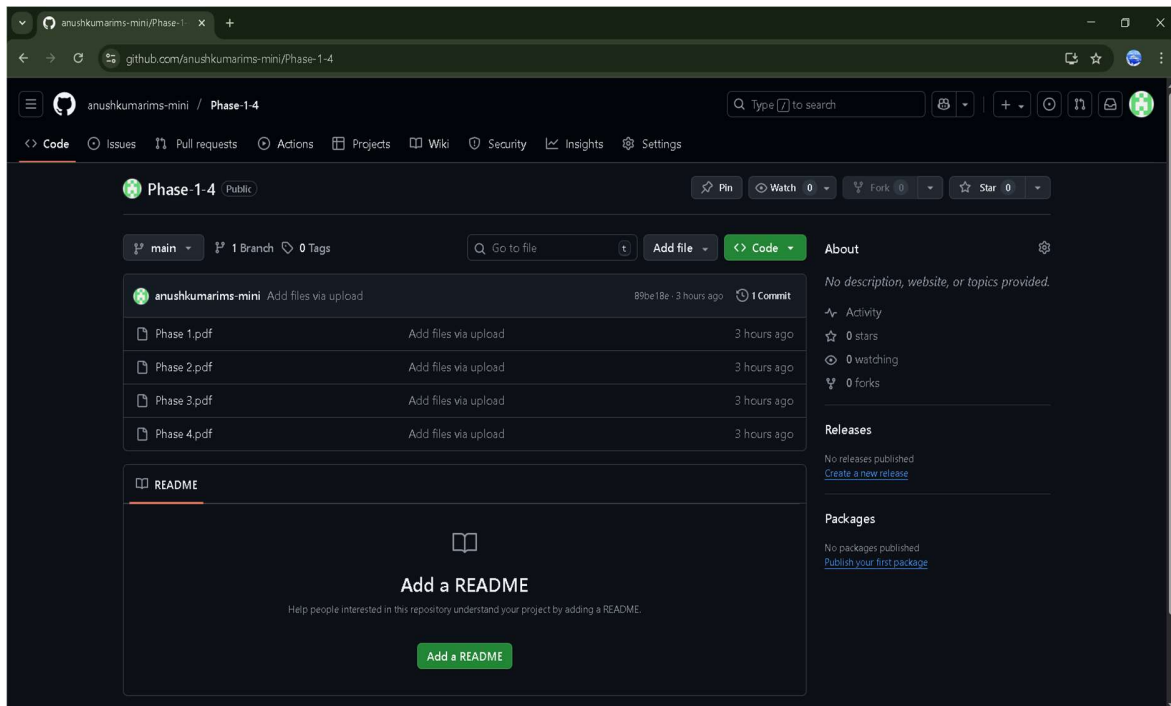


The screenshot displays a Visual Studio Code editor window. The Explorer sidebar on the left shows a project structure with folders like 'node_modules', 'public', and 'src', and files like 'index.html', 'package.json', and 'vite.config.ts'. The main editor area shows the content of 'index.html', which is an HTML document with a head section containing meta tags for charset, viewport, title, description, author, and Open Graph/Twitter metadata. The body section contains a root div and a script tag for 'src/main.tsx'. The terminal window at the bottom shows the command 'npm run dev' being executed, resulting in the Vite dev server starting on port 8080. The status bar at the bottom indicates the current file is 'index.html' at line 1, column 1, with a UTF-8 encoding and LF line endings.

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>Invalid Form Validation | Advanced Form Validation System</title>
7   <meta name="description" content="Experience real-time form validation with beautiful error states, gradient animations, and backend integration" />
8   <meta name="author" content="Form Validator Pro" />
9
10  <meta property="og:title" content="Invalid Form Validation | Advanced Form Validation System" />
11  <meta property="og:description" content="Experience real-time form validation with beautiful error states, gradient animations, and backend inte" />
12  <meta property="og:type" content="website" />
13  <meta property="og:image" content="https://lovable.dev/opengraph-image-p98pgg.png" />
14
15  <meta name="twitter:card" content="summary_large_image" />
16  <meta name="twitter:site" content="@lovable_dev" />
17  <meta name="twitter:image" content="https://lovable.dev/opengraph-image-p98pgg.png" />
18 </head>
19
20 <body>
21   <div id="root"></div>
22   <script type="module" src="/src/main.tsx"></script>
23 </body>
24 </html>
25
```

```
PS C:\Users\nexusari\Downloads\violet-form-flare-main\phase 5 invalid form validation nm project> npm run dev
VITE v5.4.19 ready in 532 ms
  → Local:   http://localhost:8080/
  → Network:  http://192.168.1.6:8080/
  → press h + enter to show help
```





Challenges & Solutions

During the development of the Interactive Form Validation project, several challenges arose that tested both the technical and problem-solving skills involved in web development. One of the major challenges was maintaining consistency between frontend and backend validation. Initially, validation rules written separately in React and Node.js often conflicted, leading to errors. This issue was solved by adopting a uniform validation structure using Yup in the frontend and Joi in the backend, ensuring both layers followed the same logic.

Another difficulty was handling incorrect user inputs without breaking the form flow. Early versions of the system reloaded or froze when invalid data was entered. To overcome this, the form was improved by using JavaScript's `preventDefault()` method along with inline error messages, allowing users to correct their mistakes without interrupting the form submission process.

Password handling also posed a challenge, especially when verifying password strength and matching confirmation fields. The solution was to include a password strength meter and an automatic check for confirmpassword matching before enabling submission.

Database connectivity issues were another obstacle during testing, as the connection often failed due to configuration errors. This was resolved by using environment variables stored securely in a .env file, which ensured reliable and secure database access across different environments. Managing multiple updates and code versions was also challenging as the project expanded. This was addressed through a Git branching strategy, using separate branches like main, dev, and feature/form-validation to keep the workflow organized and prevent conflicts.

Performance optimization became necessary when real-time validation scripts caused slight UI lag. To fix this, JavaScript logic was optimized, unnecessary re-renders were reduced, and load testing was conducted to verify smooth performance.

Preventing spam submissions and ensuring security were key priorities. Initially, the form could be exploited by bots, so CAPTCHA verification and rate limiting middleware were added to block automated requests. Crossbrowser inconsistencies were another concern, as the layout appeared different on various devices. This was resolved using Tailwind CSS, which ensured responsiveness and visual consistency. Finally, deployment challenges occurred when APIs behaved differently on the cloud server compared to local testing. These issues were debugged using Postman and fixed by adjusting routes and redeploying the project. Through these steps, the system evolved into a stable, user-friendly, and secure validation platform ready for real-world deployment.

GitHub Link

<https://github.com/anushkumarims-mini/Phase-1-4>