COLLEGE CODE: 9111

COLLEGE NAME : SRM MADURAI COLLEGE FOR ENGINEERING AND TECHNOLOGY

DEPARTMENT: BTECH INFORMATION TECHNOLOGY

STUDENT NM-ID: CD7664F3F8C9B49AC4A4D39E764A810D
ROLL NO: 23IT38
REGISTER NUMBER : 911123205040
DATE: 27-09-2025

PHASE 4 : Front end technologies : Enhancements & Deployment

PROJECT NAME : Interactive form validation

Submitted by :

M. PRIYA DHARSHINI
8778555137

1. Additional Features :

Content:

Form validation can go beyond basic required field checks and format validation by adding features like:

- **Conditional Validation:** Fields that appear based on previous answers.
- **Real-time Validation:** Instant feedback while typing.
- **Custom Validation Rules:** Specific rules like password strength, age limits.
- **Multi-step Forms:** Validating step-by-step for complex forms.
- **Localization:** Display error messages in different languages.

Example Code (React with real-time validation):

```
import React, { useState } from 'react';

const FormWithAdditionalFeatures = () => {
  const [email, setEmail] = useState('');
  const [errors, setErrors] = useState({});

  const validateEmail = (value) => {
    if (!value) return "Email is required.";
    const regex = /\S+@\S+\.\S+/;
    if (!regex.test(value)) return "Invalid email format.";
    return "";
  };

  const handleChange = (e) => {
    setEmail(e.target.value);
    setErrors({ email: validateEmail(e.target.value) });
  };

  return (
    <form>
      <input
        type="email"
        value={email}
        onChange={handleChange}
```

```
        placeholder="Enter your email"
      />
      {errors.email && <p style={{color: 'red'}}>{errors.email}</p>}
      <button disabled={!!errors.email}>Submit</button>
    </form>
  );
};

export default FormWithAdditionalFeatures;
```

2. UI/UX Improvements :

Content:

Improving user experience with form validation is crucial. Some tips:

- Highlight invalid fields with color changes or icons.
- Provide clear, concise error messages.
- Use inline validation to reduce frustration.
- Animate error messages smoothly.
- Use tooltips or helper texts for guidance.
- Keyboard accessibility and screen reader support.

Example: CSS for error highlighting and tooltip

```
input:invalid {
  border-color: red;
}

.error-tooltip {
  color: red;
  font-size: 0.8em;
  margin-top: 4px;
}
```

React snippet with UI feedback:

```
const FormWithUIUX = () => {
  const [username, setUsername] = useState('');
  const [touched, setTouched] = useState(false);

  const error = username.length < 3 ? "Username must be at least 3 characters." :
"";
```

```
  return (
    <form>
      <label>Username</label>
      <input
        value={username}
        onChange={(e) => setUsername(e.target.value)}
        onBlur={() => setTouched(true)}
        style={{ borderColor: error && touched ? 'red' : 'black' }}
      />
      {error && touched && <div className="error-tooltip">{error}</div>}
      <button disabled={!!error}>Submit</button>
    </form>
  );
};
```

3. API Enhancement:

Content:

Forms often submit data to APIs. Enhancing APIs with validation helps secure backend and improve user experience:

- Server-side validation as backup to client validation.
- Return structured validation errors (JSON with field-specific errors).
- Rate limiting to prevent abuse.
- Use JSON Schema for standard validation.
- Return success messages and helpful error codes.

Node.js Express API example with validation using express-validator:

```
const express = require('express');
const { body, validationResult } = require('express-validator');

const app = express();
app.use(express.json());

app.post('/submit-form', [
  body('email').isEmail().withMessage('Must be a valid email'),
  body('password').isLength({ min: 6 }).withMessage('Password must be 6+ chars'),
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
```

```
    return res.status(400).json({ errors: errors.array() });
  }
  res.json({ message: 'Form submitted successfully!' });
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

4. Performance & Security Checks

Content:

Validation affects performance and security. Points to consider:

- **Performance:**
    - Minimize synchronous validation blocking UI.
    - Debounce real-time validations.
    - Lazy load heavy validation libraries.
- **Security:**
    - Never trust client-side validation alone.
    - Protect against injection attacks (SQL, XSS).
    - Use HTTPS to encrypt data in transit.
    - Sanitize inputs before database entry.
    - Limit API requests to prevent brute force.

Debouncing validation example:

```
function debounce(fn, delay) {
  let timer;
  return (...args) => {
    clearTimeout(timer);
    timer = setTimeout(() => fn(...args), delay);
  };
}

// Usage inside an input handler:
const handleEmailChange = debounce((value) => {
  // validate email after user stops typing for 500ms
  validateEmail(value);
}, 500);
```

5. Testing of Enhancements :

Content:

Testing validation is critical:

- Unit tests for validation functions.
- Integration tests for form submission.
- Use tools like Jest, React Testing Library.
- Mock API calls in tests.
- E2E testing with Cypress or Selenium.
- Test edge cases (empty fields, invalid formats).

Jest example testing validation function:

```
function isEmailValid(email) {
  const regex = /\S+@\S+\.\S+/;
  return regex.test(email);
}

test('validates correct email', () => {
  expect(isEmailValid('test@example.com')).toBe(true);
});

test('rejects invalid email', () => {
  expect(isEmailValid('test@')).toBe(false);
});
```

6. Deployment (Netlify, Vercel, or Cloud Platform)

Content:

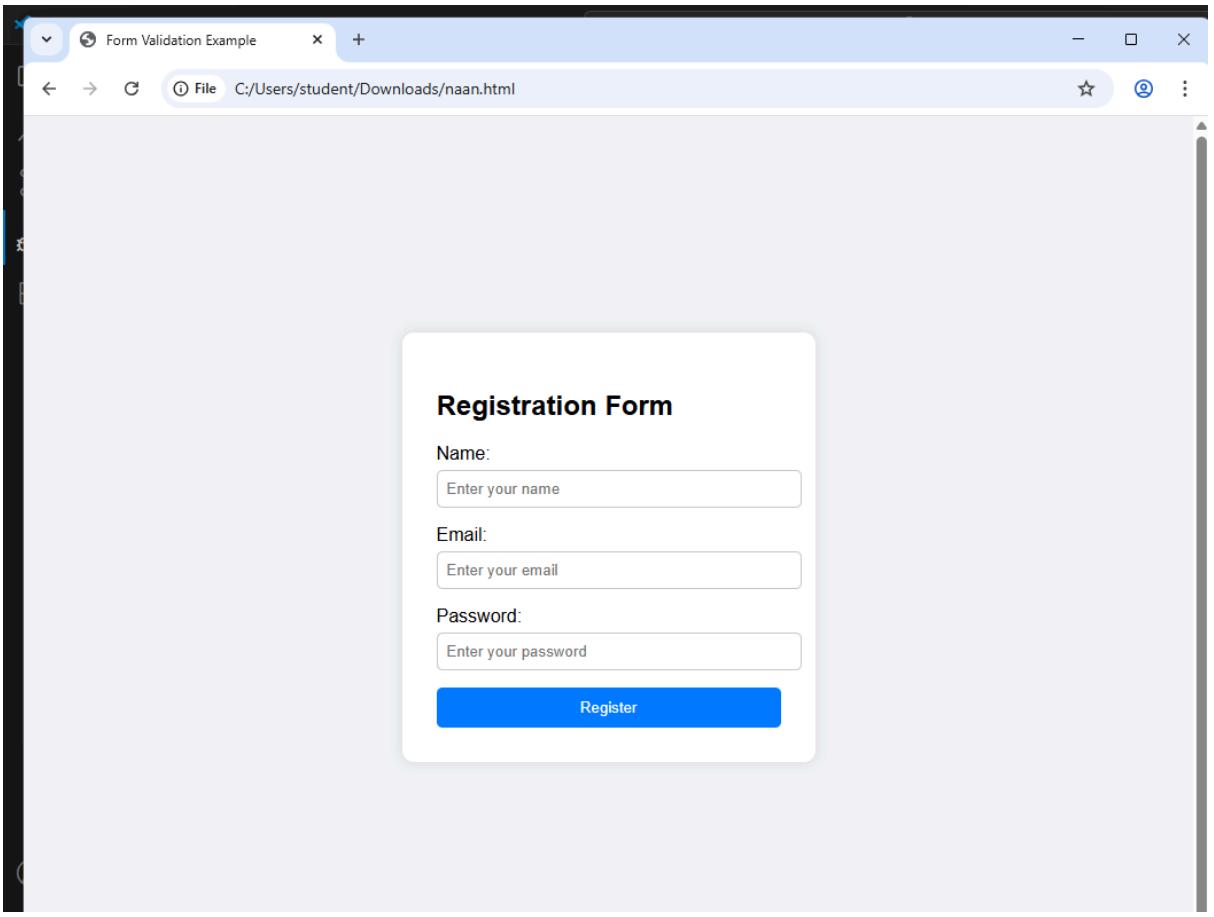
After developing the form with validation, deployment is the last step.

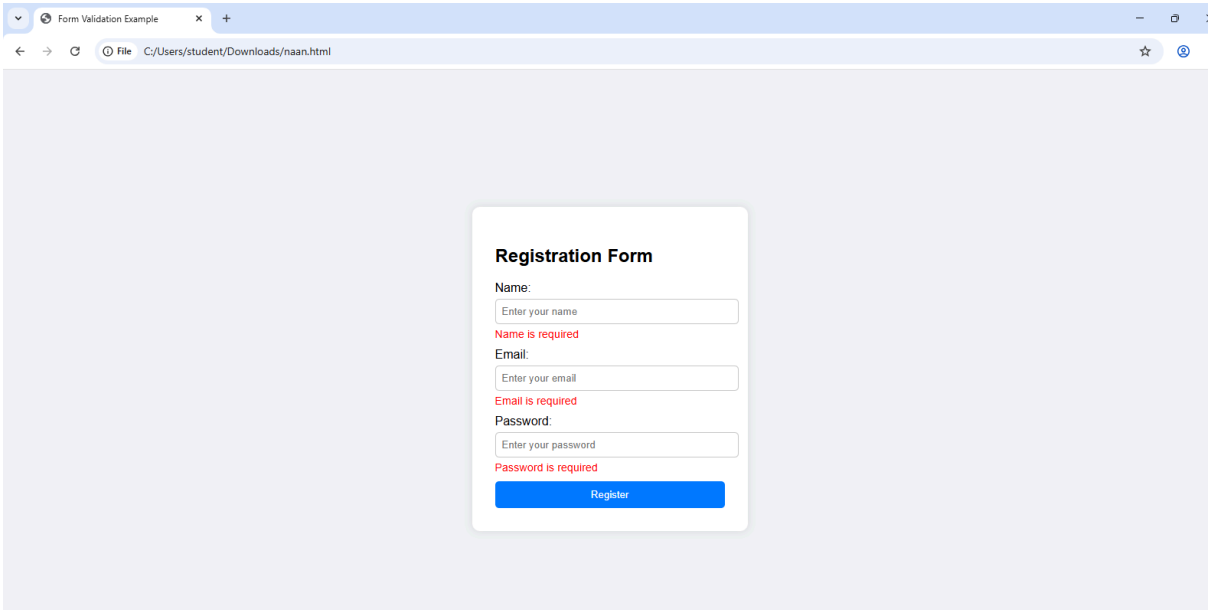
- **Netlify/Vercel:** Great for frontend apps (React, Vue).
- Connect GitHub repo, auto-deploy on push.
- Environment variables to store API keys.
- Use serverless functions (Netlify Functions, Vercel Serverless) for backend API.
- Cloud platforms (AWS, GCP, Azure) for scalable backend and databases.
- Enable HTTPS and configure domain.

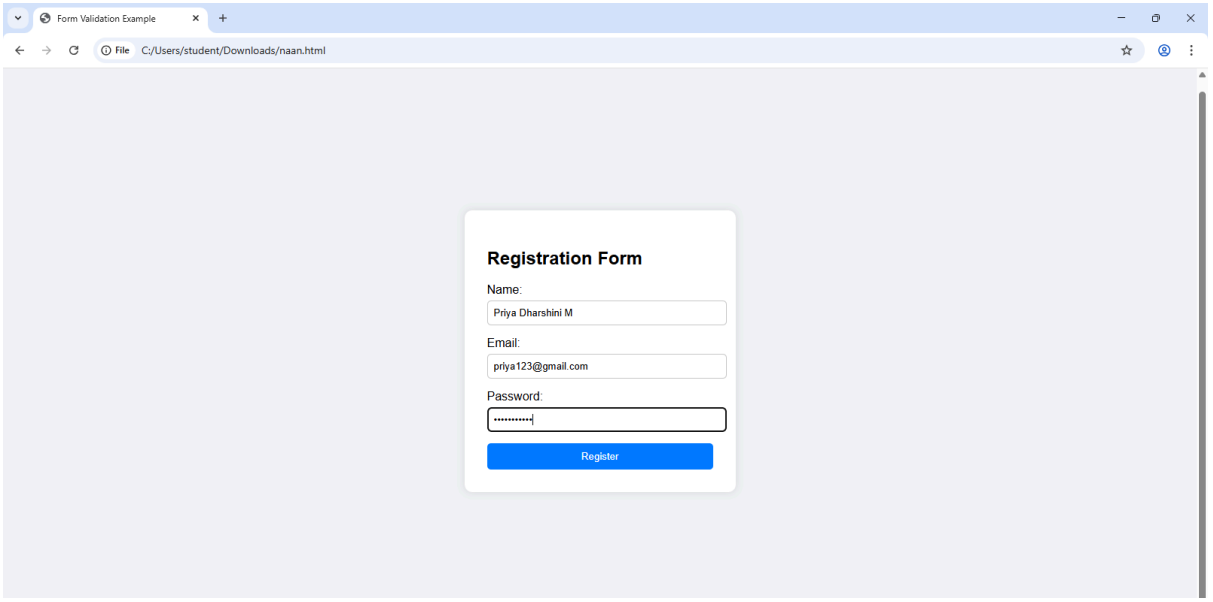
Example: Deploy React app to Netlify

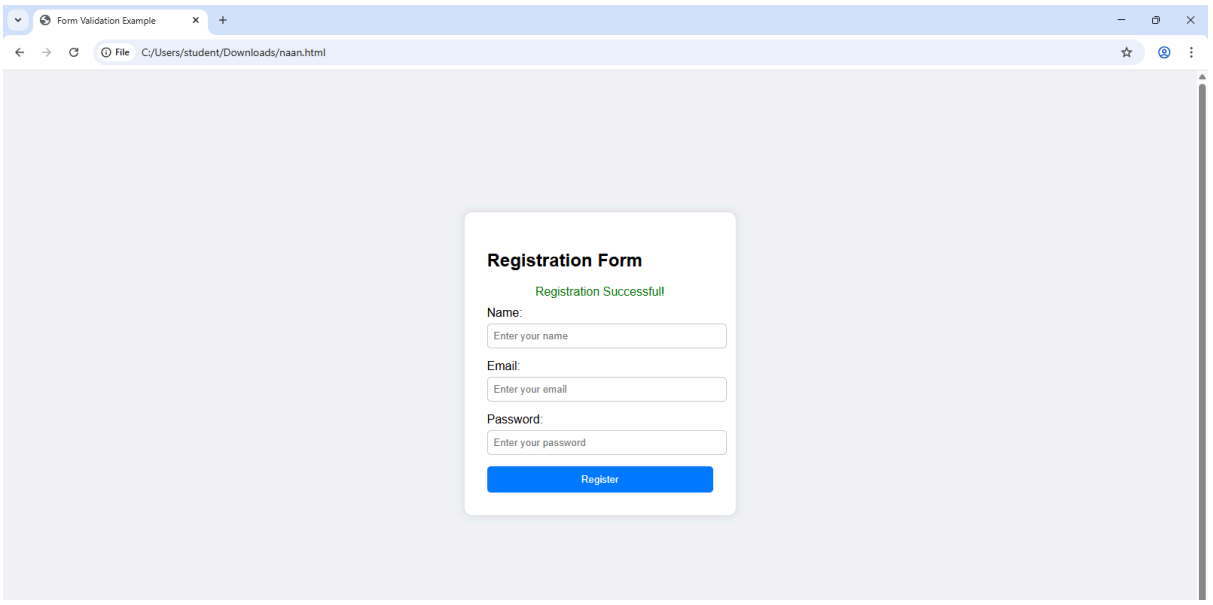
1. Push your React project to GitHub.
2. Go to Netlify, create a new site.
3. Connect your GitHub repo.
4. Set build command: npm run build
5. Publish directory: build
6. Deploy and get live URL.

OUTPUT :




Form Validation Example
File C:/Users/student/Downloads/naan.html
Registration Form
Name:
Enter your name
Name is required
Email:
Enter your email
Email is required
Password:
Enter your password
Password is required
Register


Form Validation Example
File C:/Users/student/Downloads/naan.html
Registration Form
Name:
Priya Dharshini M
Email:
priya123@gmail.com
Password:
Register


Form Validation Example
File C:/Users/student/Downloads/naan.html
Registration Form
Registration Successful!
Name:
Enter your name
Email:
Enter your email
Password:
Enter your password
Register

GIT HUB LINK :
https://github.com/priyadharshinimanikandan25-art/Interactive-Form-Validation.git