Here's a simple form validation project in React that includes validation for different types of inputs like text, email, password, and more. This example demonstrates how to handle form input, validate data, and display error messages.

Form Validation Project

Project Structure:

- src
  - components
    - FormValidation.js
  - App.js
  - index.js

FormValidation.js

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

function FormValidation() {
  const [formValues, setFormValues] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: '',
  });

  const [formErrors, setFormErrors] = useState({});
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormValues({ ...formValues, [name]: value });
  };

  const validate = () => {
    let errors = {};

    // Name validation
    if (!formValues.name.trim()) {
      errors.name = 'Name is required';
    }
```

```
  // Email validation
  if (!formValues.email) {
    errors.email = 'Email is required';
  } else if (!/\S+@\S+\.\S+/.test(formValues.email)) {
    errors.email = 'Email address is invalid';
  }

  // Password validation
  if (!formValues.password) {
    errors.password = 'Password is required';
  } else if (formValues.password.length < 6) {
    errors.password = 'Password needs to be 6 characters or more';
  }

  // Confirm password validation
  if (!formValues.confirmPassword) {
    errors.confirmPassword = 'Confirm password is required';
  } else if (formValues.confirmPassword !== formValues.password) {
    errors.confirmPassword = 'Passwords do not match';
  }

  return errors;
};

const handleSubmit = (e) => {
  e.preventDefault();
  setFormErrors(validate());
  setIsSubmitting(true);
};

// Simulate form submission
React.useEffect(() => {
  if (Object.keys(formErrors).length === 0 && isSubmitting) {
    console.log('Form submitted successfully', formValues);
    // Here you can make API calls or further processing
    alert('Form submitted successfully!');
  }
}, [formErrors, isSubmitting, formValues]);

return (
  <div className="form-container">
    <form onSubmit={handleSubmit} noValidate>
```

```
      <h1>Form Validation Example</h1>

      <div className="form-group">
        <label htmlFor="name">Name</label>
        <input
          type="text"
          name="name"
          id="name"
          value={formValues.name}
          onChange={handleChange}
        />
        {formErrors.name && <span
className="error">{formErrors.name}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="email">Email</label>
        <input
          type="email"
          name="email"
          id="email"
          value={formValues.email}
          onChange={handleChange}
        />
        {formErrors.email && <span
className="error">{formErrors.email}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="password">Password</label>
        <input
          type="password"
          name="password"
          id="password"
          value={formValues.password}
          onChange={handleChange}
        />
        {formErrors.password && <span
className="error">{formErrors.password}</span>}
      </div>

      <div className="form-group">
        <label htmlFor="confirmPassword">Confirm Password</label>
```

```
          <input
            type="password"
            name="confirmPassword"
            id="confirmPassword"
            value={formValues.confirmPassword}
            onChange={handleChange}
          />
          {formErrors.confirmPassword && <span
className="error">{formErrors.confirmPassword}</span>}
        </div>

        <button type="submit">Submit</button>
      </form>
    </div>
  );
}

export default FormValidation;
```

App.js

```
import React from 'react';
import FormValidation from './components/FormValidation';
import './App.css';

function App() {
  return (
    <div className="App">
      <FormValidation />
    </div>
  );
}

export default App;
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

index.css (optional for styling)

```
css

body {
  font-family: Arial, sans-serif;
}

.form-container {
  max-width: 400px;
  margin: 2rem auto;
  padding: 1rem;
  border: 1px solid #ccc;
  border-radius: 4px;
  background-color: #f9f9f9;
}

.form-group {
  margin-bottom: 1rem;
}

label {
  display: block;
  margin-bottom: 0.5rem;
}

input {
  width: 100%;
  padding: 0.5rem;
```

```
  margin-bottom: 0.5rem;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.error {
  color: red;
  font-size: 0.875rem;
}

button {
  padding: 0.5rem 1rem;
  border: none;
  border-radius: 4px;
  background-color: #007bff;
  color: #fff;
  cursor: pointer;
}

button:hover {
  background-color: #0056b3;
}
```

Explanation

1. **State Management**:
   - The form's input values are managed using the useState hook.
   - formErrors stores validation errors.
2. **Validation Logic**:
   - The validate function checks each field for specific conditions (e.g., email format, password length) and returns an object with error messages if any validations fail.
3. **Form Submission**:
   - handleSubmit prevents the default form submission, triggers validation, and sets isSubmitting to true.
   - If there are no errors, the form data is logged (or further processed, like sending to an API).
4. **Effect Hook**:

- The useEffect hook monitors formErrors and isSubmitting to handle form submission logic.

5 **Error Display**:

- Error messages are displayed conditionally based on the presence of error messages in the formErrorsobject.

This example demonstrates a typical form validation scenario in a React application. You can expand this project by adding more input fields, enhancing the UI, or integrating with an API for backend validation.