<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta

name="description"

content="Web site created using create-react-app"

/>

<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />

<!--

manifest.json provides metadata used when your web app is installed on a

user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/

-->

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

<!--

Notice the use of %PUBLIC\_URL% in the tags above.

It will be replaced with the URL of the `public` folder during the build.

Only files inside the `public` folder can be referenced from the HTML.

Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will

work correctly both with client-side routing and a non-root public URL.

Learn how to configure a non-root public URL by running `npm run build`.

-->

<title>React App</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

<!--

This HTML file is a template.

If you open it directly in the browser, you will see an empty page.

You can add webfonts, meta tags, or analytics to this file.

The build step will place the bundled scripts into the <body> tag.

To begin the development, run `npm start` or `yarn start`.

To create a production bundle, use `npm run build` or `yarn build`.

-->

</body>

</html>

{

"short\_name": "React App",

"name": "Create React App Sample",

"icons": [

{

"src": "favicon.ico",

"sizes": "64x64 32x32 24x24 16x16",

"type": "image/x-icon"

},

{

"src": "logo192.png",

"type": "image/png",

"sizes": "192x192"

},

{

"src": "logo512.png",

"type": "image/png",

"sizes": "512x512"

}

],

"start\_url": ".",

"display": "standalone",

"theme\_color": "#000000",

"background\_color": "#ffffff"

}

.App {

font-family: Arial, sans-serif;

text-align: center;

padding: 30px;

}

input {

width: 100%;

padding: 8px;

margin-top: 5px;

}

button {

padding: 10px 20px;

margin-top: 10px;

}

import React from 'react';

import './App.css';

import Register from './Register';

function App() {

return (

<div className="App">

<h2>Mail Registration Form</h2>

<Register />

</div>

);

}

export default App;

import { render, screen } from '@testing-library/react';

import App from './App';

test('renders learn react link', () => {

render(<App />);

const linkElement = screen.getByText(/learn react/i);

expect(linkElement).toBeInTheDocument();

});

import React, { useState } from 'react';

function Register() {

const [formData, setFormData] = useState({

name: '',

email: '',

password: '',

});

const [errors, setErrors] = useState({});

// 🔄 Handle Input Changes

const handleChange = (e) => {

const { name, value } = e.target;

setFormData((prev) => ({

...prev,

[name]: value,

}));

};

// ✅ Handle Form Submit

const handleSubmit = (e) => {

e.preventDefault();

const newErrors = {};

if (formData.name.length < 5) {

newErrors.name = 'Name should be at least 5 characters.';

}

if (!formData.email.includes('@') || !formData.email.includes('.')) {

newErrors.email = 'Email must include @ and .';

}

if (formData.password.length < 8) {

newErrors.password = 'Password must be at least 8 characters.';

}

setErrors(newErrors);

if (Object.keys(newErrors).length === 0) {

alert('Registration successful!');

setFormData({ name: '', email: '', password: '' });

}

};

return (

<form onSubmit={handleSubmit} style={{ width: '400px', margin: 'auto' }}>

<div style={{ marginBottom: '10px' }}>

<label>Name:</label><br />

<input

type="text"

name="name"

value={formData.name}

onChange={handleChange}

/>

{errors.name && <div style={{ color: 'red' }}>{errors.name}</div>}

</div>

<div style={{ marginBottom: '10px' }}>

<label>Email:</label><br />

<input

type="text"

name="email"

value={formData.email}

onChange={handleChange}

/>

{errors.email && <div style={{ color: 'red' }}>{errors.email}</div>}

</div>

<div style={{ marginBottom: '10px' }}>

<label>Password:</label><br />

<input

type="password"

name="password"

value={formData.password}

onChange={handleChange}

/>

{errors.password && <div style={{ color: 'red' }}>{errors.password}</div>}

</div>

<button type="submit">Register</button>

</form>

);

}

export default Register;

body {

margin: 0;

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

sans-serif;

-webkit-font-smoothing: antialiased;

-moz-osx-font-smoothing: grayscale;

}

code {

font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

monospace;

}

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

const reportWebVitals = onPerfEntry => {

if (onPerfEntry && onPerfEntry instanceof Function) {

import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {

getCLS(onPerfEntry);

getFID(onPerfEntry);

getFCP(onPerfEntry);

getLCP(onPerfEntry);

getTTFB(onPerfEntry);

});

}

};

export default reportWebVitals;

// jest-dom adds custom jest matchers for asserting on DOM nodes.

// allows you to do things like:

// expect(element).toHaveTextContent(/react/i)

// learn more: https://github.com/testing-library/jest-dom

import '@testing-library/jest-dom';

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated