**1. React Router (for Navigation)**

**Setting up React Router**

* Install React Router:

bash

Copy code

npm install react-router-dom

* Import BrowserRouter (or HashRouter for hash-based URLs) and wrap the app:

jsx

Copy code

import { BrowserRouter as Router } from 'react-router-dom';

function App() {

return (

<Router>

<div>App Content</div>

</Router>

);

}

**Route Configuration (<Route>, <Switch>, <Link>, <NavLink>):**

1. **<Route>**:
   * Defines a path and the component to render.
   * Example:

jsx

Copy code

import { Route } from 'react-router-dom';

function App() {

return (

<Route path="/home" component={HomePage} />

);

}

1. **<Switch>**:
   * Ensures only the first matching route is rendered.
   * Example:

jsx

Copy code

import { Switch, Route } from 'react-router-dom';

function App() {

return (

<Switch>

<Route path="/home" component={HomePage} />

<Route path="/about" component={AboutPage} />

</Switch>

);

}

1. **<Link>**:
   * Navigates between routes without refreshing the page.
   * Example:

jsx

Copy code

import { Link } from 'react-router-dom';

function Navigation() {

return (

<div>

<Link to="/home">Home</Link>

<Link to="/about">About</Link>

</div>

);

}

1. **<NavLink>**:
   * Similar to <Link>, but allows for active styling.
   * Example:

jsx

Copy code

import { NavLink } from 'react-router-dom';

function Navigation() {

return (

<div>

<NavLink to="/home" activeClassName="active">Home</NavLink>

</div>

);

}

**Programmatic Navigation using useHistory**

* useHistory allows navigation through code.
* Example:

jsx

Copy code

import { useHistory } from 'react-router-dom';

function HomePage() {

const history = useHistory();

function navigateToAbout() {

history.push('/about');

}

return <button onClick={navigateToAbout}>Go to About</button>;

}

**Nested Routes and Route Parameters**

1. **Nested Routes**:
   * Example:

jsx

Copy code

function Dashboard() {

return (

<Switch>

<Route path="/dashboard/profile" component={Profile} />

<Route path="/dashboard/settings" component={Settings} />

</Switch>

);

}

1. **Route Parameters**:
   * Example:

jsx

Copy code

function UserPage({ match }) {

return <h1>User ID: {match.params.id}</h1>;

}

<Route path="/user/:id" component={UserPage} />;

**Handling 404 Pages**

* Use a Route with no path:

jsx

Copy code

function App() {

return (

<Switch>

<Route path="/home" component={HomePage} />

<Route path="/about" component={AboutPage} />

<Route component={NotFoundPage} />

</Switch>

);

}

**2. Forms and Input Handling**

**Controlled vs. Uncontrolled Components**

1. **Controlled Components**:
   * Form data is handled by React state.
   * Example:

jsx

Copy code

function ControlledForm() {

const [name, setName] = React.useState('');

return (

<input value={name} onChange={(e) => setName(e.target.value)} />

);

}

1. **Uncontrolled Components**:
   * Form data is handled by the DOM.
   * Example:

jsx

Copy code

function UncontrolledForm() {

const inputRef = React.useRef();

function handleSubmit() {

alert(inputRef.current.value);

}

return <input ref={inputRef} />;

}

**Form Validation**

1. **HTML5 Constraints**:
   * Use required, min, max, etc.
   * Example:

jsx

Copy code

<form>

<input type="text" required />

<input type="number" min="1" max="10" />

</form>

1. **Formik and Yup**:
   * Install libraries:

bash

Copy code

npm install formik yup

* + Example:

jsx

Copy code

import { Formik, Field, Form } from 'formik';

import \* as Yup from 'yup';

const ValidationSchema = Yup.object({

name: Yup.string().required('Required'),

email: Yup.string().email('Invalid email').required('Required'),

});

function SignupForm() {

return (

<Formik

initialValues={{ name: '', email: '' }}

validationSchema={ValidationSchema}

onSubmit={(values) => console.log(values)}

>

{({ errors, touched }) => (

<Form>

<Field name="name" />

{errors.name && touched.name && <div>{errors.name}</div>}

<Field name="email" />

{errors.email && touched.email && <div>{errors.email}</div>}

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

</Form>

)}

</Formik>

);

}

**Handling Multiple Inputs**

* Use a single state object:

jsx

Copy code

function MultiInputForm() {

const [formData, setFormData] = React.useState({ name: '', email: '' });

function handleChange(e) {

const { name, value } = e.target;

setFormData((prev) => ({ ...prev, [name]: value }));

}

return (

<form>

<input name="name" value={formData.name} onChange={handleChange} />

<input name="email" value={formData.email} onChange={handleChange} />

</form>

);

}

**File Uploads**

* Use onChange for file input:

function FileUpload() {

function handleFileChange(e) {

const file = e.target.files[0];

console.log(file);

}

return <input type="file" onChange={handleFileChange} />;

}