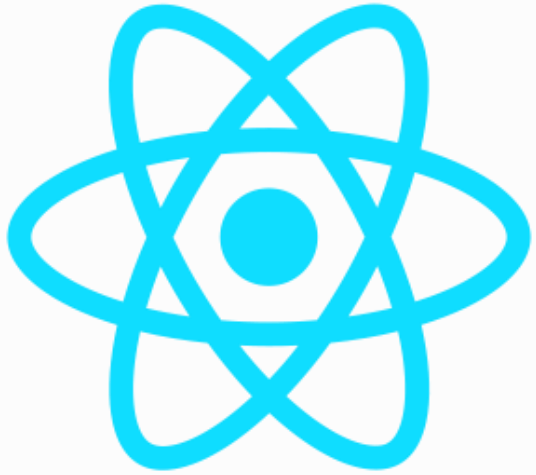
sourcemind

REACT ROUTER



React JS

React Router

React Router enables "client side routing".

In traditional websites, the browser requests a document from a web server, downloads and evaluates CSS and JavaScript assets, and renders the HTML sent from the server. When the user clicks a link, it starts the process al over again for a new page.

Client side routing allows your app to update the URL from a link click without making another request for another docu from the server. Instead, your app can immediately render some new UI and make data requests with fetch to update the page with new information.

`<BrowserRouter>`

▶ Type declaration

A `<BrowserRouter>` stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.

`<BrowserRouter window>` defaults to using the current document's `defaultView`, but it may also be used to track changes to another window's URL, in an `<iframe>`, for example.

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { BrowserRouter } from "react-router-dom";

ReactDOM.render(
  <BrowserRouter>
    {/* The rest of your app goes here */}
  </BrowserRouter>,
  root
);
```

`<HashRouter>`

▶ Type declaration

`<HashRouter>` is for use in web browsers when the URL should not (or cannot) be sent to the server for some reason. This may happen in some shared hosting scenarios where you do not have full control over the server. In these situations, `<HashRouter>` makes it possible to store the current location in the `hash` portion of the current URL, so it is never sent to the server.

`<HashRouter window>` defaults to using the current document's `defaultView`, but it may also be used to track changes to another window's URL, in an `<iframe>`, for example.

```
import * as React from "react";
import * as ReactDOM from "react-dom";
import { HashRouter } from "react-router-dom";

ReactDOM.render(
  <HashRouter>
    {/* The rest of your app goes here */}
  </HashRouter>,
  root
);
```

> ⚠ IMPORTANT
> We strongly recommend you do not use `HashRouter` unless you absolutely have to.

`<Router>`

▶ Type declaration

`<Router>` is the low-level interface that is shared by all router components (like `<BrowserRouter>` and `<StaticRouter>`). In terms of React, `<Router>` is a context provider that supplies routing information to the rest of the app.

You probably never need to render a `<Router>` manually. Instead, you should use one of the higher-level routers depending on your environment. You only ever need one router in a given app.

The `<Router basename>` prop may be used to make all routes and links in your app relative to a "base" portion of the URL pathname that they all share. This is useful when rendering only a portion of a larger app with React Router or when your app has multiple entry points. Basenames are not case-sensitive.

Usage

```
<BrowserRouter>
    <Routes>
        <Route element={<MainPage />} path="/" />
        <Route element={<LoginPage />} path="/login" />
        <Route element={<RegistrationPage />} path="/register" />
        <Route element={<ItemDetails />} path="/item/:id" />
        <Route path="*" element={<NotFoundPage />} />
    </Routes>
</BrowserRouter>
```

Link

A <Link> is an element that lets the user navigate to another page by clicking or tapping on it. In react-router-dom, a <Link> renders an accessible <a> element with a real href that points to the resource it's linking to. This means that things like right-clicking a <Link> work as you'd expect. You can use <Link reloadDocument> to skip client side routing and let the browser handle the transition normally (as if it were an <a href>).

```
import * as React from "react";
import { Link } from "react-router-dom";

function UsersIndexPage({ users }) {
  return (
    <div>
      <h1>Users</h1>
      <ul>
        {users.map((user) => (
          <li key={user.id}>
            <Link to={user.id}>{user.name}</Link>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

NavLink

A <NavLink> is a special kind of <Link> that knows whether or not it is "active". This is useful when building a navigation menu such as a breadcrumb or a set of tabs where you'd like to show which of them is currently selected. It also provides useful context for assistive technology like screen readers.

```
  <NavLink
    to="messages"
    style={({ isActive }) =>
      isActive ? activeStyle : undefined
    }
  >
    Messages
  </NavLink>
</li>
<li>
  <NavLink
    to="tasks"
    className={({ isActive }) =>
      isActive ? activeClassName : undefined
    }
  >
    Tasks
  </NavLink>
```