Routing

Routing is a process in which a browser will execute different javascript file’s functional/class components based on user’s action or request**. ReactJS Router is mainly used for developing Single Page Web Applications**. React Router is used to create multiple routes in the application. When a user types a specific web page’s address also known as URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route means specific JavaScript will execute and rendered(displayed) on screen.

React Router dom is a library used to create routes in the React application using React Router Package. Without React Router, **it is not possible to display multiple views in React applications.** Most of the social media websites like Facebook, Instagram uses React Router for rendering (displaying) multiple views. It is not possible to use react-router directly in your application. To use react routing, first, you need to install **react-router-dom modules** in your application. Run below command to install react-router-dom package.

```
npm install react-router-dom
```

Router Components in React Router

React Router provides two main types of routers: `BrowserRouter` and `HashRouter`. The choice between them depends on your hosting environment and how you want your routes to behave.

1. **BrowserRouter:**
    - Uses standard URL paths (e.g., `/about`) for routing.
    - **Typically used when you have server-side rendering or you are hosting your app on a web server that can handle dynamic routes.**
    - It's more suitable for production applications with a configured server.
    - Requires server configuration to handle dynamic routes.

2. **HashRouter:**
    - Uses the URL hash (e.g., `/#/about`) for routing.
    - **Suitable for hosting in environments where configuring the server for dynamic routes is not an option (e.g., static sites, GitHub Pages).**
    - The hash portion of the URL is entirely client-side, so it doesn't require server configuration.

Example of BrowserRouter

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter, Route, Routes } from 'react-router-dom';
let Home = () => <div><h1> Home Page </h1> </div>;
let Aboutus = () => <div><h1> About Us </h1> </div>;
function App() {
    return (
        <BrowserRouter>
            <Routes>
                <Route index path="/" element={<Home />} />
                <Route path="/aboutus" element={<Aboutus />} />
            </Routes>
        </BrowserRouter>
    );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />)
```

Example of HashRouter

```
import React from 'react';
import { HashRouter, Route, Link } from 'react-router-dom';

const Home = () =><div>Home Page</div>;
const About = () =><div>About Page</div>;

function App() {
  return (
<HashRouter>
<div>
<ul>
<li>
<Link to="/">Home</Link>
</li>
<li>
<Link to="/about">About</Link>
</li>
</ul>

<hr />

<Route exact path="/" component={Home} />
<Route path="/about" component={About} />
</div>
</HashRouter>
  );
}
export default App;
```

now let us see one real world example.layout.js (menu)

```
import { Outlet, Link } from"react-router-dom";
var Layout = () => {
  return (
    <div>
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/blogs">Blogs</Link>
          </li>
          <li> /*dynamic routing */
           <Link to="/contact/9662512857/ankit3385@gmail.com">Contact</Link>

</li>

        </ul>
      </nav>
      <Outlet />
    </div>
  )
};
Export default Layout;
```

Home.js

```
export default function Home() {
    return<h1>Home</h1>;
  };
```

Blogs.js

```
export default function Blog() {
    return<h1>Blog Articles</h1>;
  };
```

Contact.js

```
import { useParams } from"react-router-dom"
export default function Contact() {
  let { mobile,email } = useParams() /* useParam hook has parameters */
  return (
    <div>
      <h1>Contact No: -{mobile}</h1>
      <h2>Emailaddress: -{email}</h2>
    </div>
  )
}
```

NoPage.js

```
export default function PageNotFound {
    return <h1>404 – Such Page does not exists,</h1>;
  };
```

Index.js

```
import ReactDOM from"react-dom/client";
import { BrowserRouter, Routes, Route } from"react-router-dom";
import Layout from"./Layout";
import Home from"./Home";
import Blogs from"./Blogs";
import Contact from"./Contact";
import NoPage from"./NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout/>}>
          <Route index element={<Home/>}/>
          <Route path="blogs" element={<Blogs/>}/>
         <Route path="contact/:mobile/:email" element={<Contact/>} /> /* dynamic rounting */

          <Route path="*" element={<NoPage/>} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App/>);
```

Now let us understand first

1. We wrap our content first with <BrowserRouter>.
2. Then we define our <Routes>. An application can have multiple <Routes>.
3. <Route>s can be nested. The first <Route> has a path of / and renders the Layout component.
4. The nested <Route>s inherit and add to the parent route. So the blogs path is combined with the parent and becomes /blogs.
5. The Home component route does not have a path but has an index attribute. That specifies this route as the default route for the parent route, which is /.
6. Setting the `path` to `*` will act as a catch-all for any undefined URLs. This is great for a 404 error page.

7. The Layout component has <Outlet> and <Link> elements.
8. The <Outlet> renders the current route selected.
9. <Link> is used to set the URL and keep track of browsing history.
10. Anytime we link to an internal path, we will use <Link> instead of <a href="">.
11. The "layout route" is a shared component that inserts common content on all pages, such as a navigation menu.
12. We also have some value passed in Contact Route mainly mobile and email address. Which will be passed to Routes and can be accessed in Contact.js file using use useParam hook. the `useParams` hook takes no parameters and will return an object with keys that match the dynamic parameters in your route. In our case it will return mobile and email address.

Create Routes with priority

When we were just dealing with hard coded routes it was pretty easy to know which route would be rendered, but when dealing with dynamic routes it can be a bit more complicated. Take these routes for example.

```
<Routes>
<Route path="/" element={<Home />} />
<Route path="/books" element={<BookList />} />
<Route path="/books/:id" element={<Book />} />
<Route path="/books/new" element={<NewBook />} />
</Routes>
```

React Router will use an algorithm to decide which route is most likely the one you want to open. In our case react will render the

1. `BookList in` case of books/.
2. `Book in` case of books/:id.
3. NewBook `in` case of books/new.

The actual way this algorithm works is very similar to CSS specificity since it will try to determine which route that matches our URL is the most specific (has the least amount of dynamic elements) and it will choose that route.

Book.js

```
function Book(){
    return (
        <h1>This is Book with id </h1>
    )
}
exportdefault Book;
```

NewBook.js

```
function NewBook(){
    return (
        <h1>This is New Book</h1>
    )
}
exportdefault NewBook;
```

BookList.js

```
function BookList(){
    return (
        <h1>This is BookList</h1>
    )
}
exportdefault BookList;
```
layout.js

```
import { Outlet, Link} from"react-router-dom";
const Layout = () => {
  return (
    <div>
      <nav>
        <ul>
          <li>
            <Linkto="/">Home</Link>
          </li>
          <li>
            <Linkto="/books/new">Add new Book</Link>
          </li>
          <li>
            <Linkto="/books">Book List</Link>
          </li>
          <li>
            <Linkto="/books/1">Book with id</Link>
          </li>
        </ul>
      </nav>
      <Outlet/>
    </div>
  )
};
exportdefault Layout;
```
index.js

```
import ReactDOM from"react-dom/client";
import { BrowserRouter, Routes, Route } from"react-router-dom";
import BookList from'./BookList.js'
import Book from'./Book'
import NewBook from'./NewBook.js'
import NoPage from"./NoPage";
import Home from"./Home";
import Layout from"./Layout";
exportdefaultfunction App() {
  return (
    <BrowserRouter>
      <Routes>
        <Routepath="/"element={<Layout/>}>
          <Routepath="/"element={<Home/>}/>
          <Routepath="/books"element={<BookList/>}/>
          <Routepath="/books/:id"element={<Book/>}/>
          <Routepath="/books/new"element={<NewBook/>}/>
          <Routepath="*"element={<NoPage/>}/>
        </Route>
      </Routes>
    </BrowserRouter>
  );
}
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
       root.render(<App/>);
```

how to redirect to any route programmatically?

To redirect to any router programmatically, we use **useNavigate** Hook.

The navigate hook is part of the React Router library, and it provides a programmatic way to navigate between different routes in a single-page React application. It is often used in functional components to trigger route changes based on user interactions or application logic.

Here's an explanation of the navigate hook:

1. **Importing the Hook:** To use the navigate hook, you need to import it from the 'react-router-dom' library:

```
import { useNavigate } from 'react-router-dom';
```

Using the `navigate` Hook: The useNavigate hook returns a function that you can call to navigate to a different route. You can use this function to programmatically change the current route within your component.

```
const navigate = useNavigate();

// Later in your component or function
navigate('/new-route');
```

Navigating to a Route: You can pass a string representing the target route as an argument to the 1navigate function. For example, navigate('/new-route') will navigate to the '/new-route' URL.

Navigating with State: You can also pass state data along with the navigation. This can be useful for passing additional information to the destination route.

```
navigate('/new-route', { name: 'Ankit' });
```

Using the `navigate` Function: The navigate function is typically called in response to user actions or within functions that are triggered by events. For example, you might use it in response to a button click or form submission.

Here's a simple example of using the navigate hook in a React component:

```
import React from 'react';
import { useNavigate } from 'react-router-dom';

function MyComponent() {
  const navigate = useNavigate();

  const redirectToAnotherPage = () => {
    navigate('/another-page');
  }

  return (
<div>
<button onClick={redirectToAnotherPage}>Go to Another Page</button>
</div>
  );
}
```

In this example, when the button is clicked, it triggers the redirectToAnotherPage function, which uses the navigate function to change the route to '/another-page'. This allows you to navigate between different parts of your single-page application without requiring a full page refresh.

The navigate hook is a powerful tool for controlling the navigation flow in your React application, and it's especially useful in single-page applications that use React Router for routing.