1. `Link`, `NavLink` và `to`

Ở bài trước ta mới chỉ cấu hình xong router, chưa có cách nào để điều hướng giữa các trang. Để làm được điều này, ta cần sử dụng `Link` hoặc `NavLink` từ thư viện `react-router-dom`.

Tại `src/component/Header.jsx`:

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

const Header = () => {
  return (
    <header>
      <div className="logo">Logo</div>
      <nav>
        <ul>
          <li>
            <Link to={"/"}>Home</Link>
          </li>
          <li>
            <Link to={"/about"}>About</Link>
          </li>
          <li>
            <Link to={"/login"}>Login</Link>
          </li>
          <li>
            <Link to={"/register"}>Register</Link>
          </li>
        </ul>
      </nav>
    </header>
  );
};

export default Header;
```

Như vậy bạn hoàn toàn có thể sử dụng `Link` để điều hướng giữa các trang mà không cần tải lại trang.

- `Link` và `NavLink` là hai component được sử dụng để điều hướng giữa các trang trong ứng dụng React. Chúng giúp bạn tạo ra các liên kết mà không cần phải tải lại trang, giữ cho trải nghiệm người dùng mượt mà hơn.
- `Link` là một component đơn giản, trong khi `NavLink` cho phép bạn thêm các lớp CSS tùy chỉnh cho liên kết đang được chọn. Điều này rất hữu ích khi bạn muốn làm nổi bật liên kết hiện tại trong menu điều hướng.
- `to` là thuộc tính của cả hai component này, cho phép bạn chỉ định đường dẫn mà bạn muốn điều hướng đến khi người dùng nhấp vào liên kết. Bạn có thể sử dụng đường dẫn tương đối hoặc tuyệt đối.

2. Nested Routes, `Outlet` và `children` route

Bạn nhận thấy rằng component `Header` của bạn vẫn chưa được sử dụng ở bất cứ đâu, bạn cần tạo một layout để sử dụng `Header` này cho tất cả các trang:

```
import { Outlet } from "react-router-dom";
import Header from "../components/Header";
import Footer from "../components/Footer";

const ClientLayout = () => {
  return (
    <div>
      <Header />
      <main>
        <Outlet />
      </main>
      <Footer />
    </div>
  );
};

export default ClientLayout;
```

Tại đây, bạn sử dụng `Outlet` để chỉ định vị trí mà các component con sẽ được hiển thị. Điều này cho phép bạn tạo ra một layout chung cho tất cả các trang trong ứng dụng của mình.

> Lưu ý: Bạn hoàn toàn có thể tạo ra các layout riêng cho từng trang nếu bạn muốn, ví dụ như `AdminLayout`, `AuthLayout`, v.v. và sử dụng chúng trong các route tương ứng.

Bây giờ để sử dụng `ClientLayout`, bạn cần thay đổi cấu hình router của mình một chút:

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import HomePage from "../pages/HomePage";
import LoginPage from "../pages/LoginPage";
import RegisterPage from "../pages/RegisterPage";
import ClientLayout from "../layouts/ClientLayout";
import AboutPage from "../pages/AboutPage";

const router = createBrowserRouter([
  {
    path: "/",
    element: <ClientLayout />,
    children: [
      { path: "/", element: <HomePage /> },
      { path: "/about", element: <AboutPage /> },
    ],
  },
  { path: "/login", element: <LoginPage /> },
  { path: "/register", element: <RegisterPage /> },
]);

export default function AppRouter() {
```

```
  return <RouterProvider router={router} />;
}
```

Tại đây, bạn đã sử dụng `ClientLayout` làm layout cho tất cả các trang con của nó. Điều này có nghĩa là tất cả các trang con sẽ được hiển thị bên trong `Outlet` của `ClientLayout`, và `Header` sẽ được hiển thị trên tất cả các trang này.

3. Page `index`

Tại sao lại cần page `index`? Bởi vì bạn có thể sử dụng `index` để định nghĩa một route mà không cần phải chỉ định đường dẫn. Điều này rất hữu ích khi bạn muốn tạo ra một trang mặc định cho một route cha. Ví dụ, bạn có thể tạo một trang mặc định cho route `/` mà không cần phải chỉ định đường dẫn. Điều này giúp bạn dễ dàng quản lý các route trong ứng dụng của mình.

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import HomePage from "../pages/HomePage";
import LoginPage from "../pages/LoginPage";
import RegisterPage from "../pages/RegisterPage";
import ClientLayout from "../layouts/ClientLayout";
import AboutPage from "../pages/AboutPage";

const router = createBrowserRouter([
  {
    index: true,
    element: <ClientLayout />,
    children: [
      { index: true, element: <HomePage /> },
      { path: "/about", element: <AboutPage /> },
    ],
  },
  { path: "/login", element: <LoginPage /> },
  { path: "/register", element: <RegisterPage /> },
]);

export default function AppRouter() {
  return <RouterProvider router={router} />;
}
```

4. `Notfound` Route

Để xử lý các route không tồn tại, bạn có thể sử dụng route `NotFound` để hiển thị một trang lỗi 404. Điều này giúp người dùng biết rằng trang họ đang tìm kiếm không tồn tại và có thể điều hướng đến các trang khác trong ứng dụng của bạn.

```
const router = createBrowserRouter([
  {
    index: true, // Route chính
    element: <ClientLayout />,
```

```
    children: [
      { index: true, element: <HomePage /> }, // Route con
      { path: "/about", element: <AboutPage /> },
    ],
  },
  { path: "/login", element: <LoginPage /> },
  { path: "/register", element: <RegisterPage /> },
  { path: "*", element: <NotFoundPage /> }, // Route NotFound
]);

export default function AppRouter() {
  return <RouterProvider router={router} />;
}
```

5. Dynamic Route và `useParams`

Để tạo ra các route động, bạn có thể sử dụng dấu `:` để chỉ định một tham số trong đường dẫn. Điều này cho phép bạn tạo ra các route mà không cần phải chỉ định đường dẫn cụ thể.

Ví dụ, bạn có thể tạo ra một route cho trang chi tiết sản phẩm với đường dẫn `/products/:id`, trong đó `:id` là tham số động mà bạn có thể sử dụng để lấy thông tin sản phẩm từ API hoặc từ dữ liệu của bạn.

Tao một file `ProductDetailPage.jsx` trong thư mục `src/pages/`:

```
import React from "react";
import { useParams } from "react-router-dom";

const ProductDetail = () => {
  const { id } = useParams();
  return <div>Trang chi tiet san pham {id}</div>;
};

export default ProductDetail;
```

Cập nhật lại Route

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <ClientLayout />,
    children: [
      { path: "/", element: <HomePage /> },
      { path: "/about", element: <AboutPage /> },
      { path: "/products/:id", element: <ProductDetail /> }, // Route động
    ],
  },
  { path: "/login", element: <LoginPage /> },
  { path: "/register", element: <RegisterPage /> },
```

```
  { path: "*", element: <NotFoundPage /> },
]);
```

Bây giờ bạn hãy thử truy cập vào đường dẫn `/products/1234` và kiểm tra kết quả.

6. `useNavigate` và `Navigate`

`useNavigate` là một hook được cung cấp bởi React Router, cho phép bạn điều hướng đến các route khác trong ứng dụng của mình mà không cần phải sử dụng `Link` hoặc `NavLink`. Điều này rất hữu ích khi bạn muốn điều hướng đến một route khác sau khi thực hiện một hành động nào đó, chẳng hạn như sau khi người dùng đăng nhập thành công.

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

const LoginPage = () => {
  const navigate = useNavigate();

  const handleLogin = () => {
    // Sau khi login thành công
    navigate("/dashboard");
  };

  return <button onClick={handleLogin}>Login</button>;
};

export default LoginPage;
```

Khác với `Link` và `NavLink`, `useNavigate` không tạo ra một liên kết mà là một hàm mà bạn có thể gọi để điều hướng đến một route khác.

`Navigate` là một component được sử dụng để điều hướng đến một route khác mà không cần phải sử dụng `Link` hoặc `NavLink`. Điều này rất hữu ích khi bạn muốn điều hướng đến một route khác mà không cần phải thực hiện một hành động nào đó.

```
import { Navigate } from "react-router-dom";

const ProtectedPage = () => {
  const isAuthenticated = false; // giả sử người dùng chưa đăng nhập

  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  return <div>Trang bảo vệ</div>;
};
```

Trong ví dụ trên, nếu người dùng chưa đăng nhập, họ sẽ được điều hướng đến trang đăng nhập. Nếu người dùng đã đăng nhập, họ sẽ thấy nội dung của trang bảo vệ.

`replace` là một thuộc tính của `Navigate`, cho phép bạn thay thế route hiện tại bằng route mới mà không cần phải thêm route mới vào lịch sử trình duyệt. Điều này rất hữu ích khi bạn muốn điều hướng đến một route khác mà không cần phải quay lại trang trước đó.