/REACT ROUTER DOM

Client side routing for React App

Ba Nguyễn (Updated v6)

/TABLE OF CONTENTS

/TABLE OF CONTENTS

01

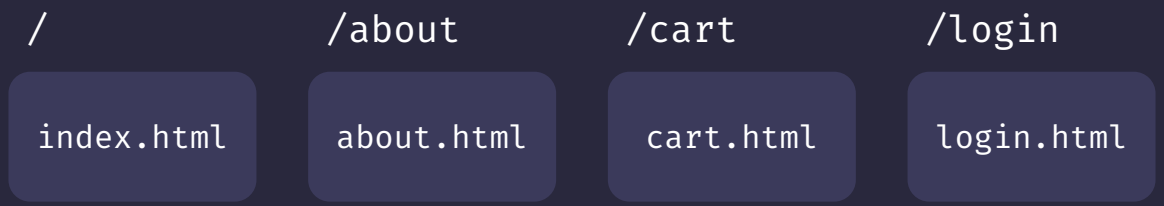
INTRODUCTION

Giới thiệu về React Router DOM

Multi-pages Application

Các trang web thông thường

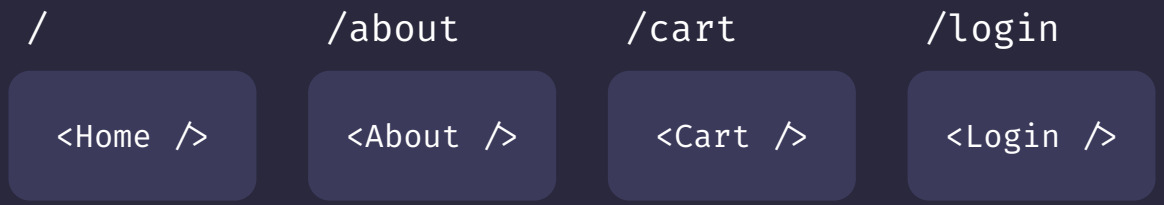
- Bao gồm nhiều trang HTML khác nhau
- Mỗi URL tương ứng với một trang
- Nội dung trang được tải từ phía server
- Tốc độ cũng như trải nghiệm người dùng kém hơn



Multi-pages Application

Các trang web sử dụng React

- Chỉ có 1 trang HTML
- Mỗi đường dẫn tương ứng với một Component (Page)
- Nội dung trang được render phía client
- Chỉ cần tải dữ liệu cần thiết (danh sách bài viết, sản phẩm, ...) để hiển thị nội dung



React Router

React Router là thư viện quản lý chế độ xem trong ứng dụng React (client side routing)

- Ẩn/hiện page (component) theo URL hiện tại
- Quản lý và cập nhật URL để thay đổi page hiển thị
- Cung cấp trải nghiệm người dùng tốt hơn
- Bao gồm 3 package riêng biệt
- Hỗ trợ 2 nền tảng Web và Mobile (React Native)

Trang chủ: https://reactrouter.com/

React Router

`react-router`

Base package bao gồm các chức năng chính, 2 packages khác dựa trên package này

`react-router-dom`

Package sử dụng cho nền tảng trình duyệt

`react-router-native`

Package sử dụng cho nền tảng di động

02

BASIC

Sử dụng React Router DOM trong ứng dụng React

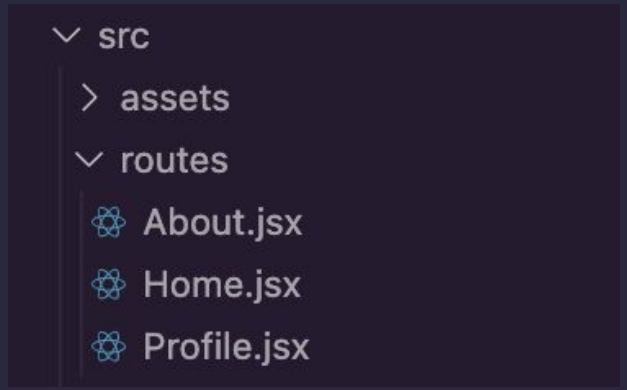
Step 1: Install

- Tạo project React mới (sử dụng CRA, Vite, ...)
- Cài đặt thư viện `react-router-dom`

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

Step 2: Creating pages

- Tạo thư mục `src/routes`
- Tạo các trang `Home.jsx`, `About.jsx`, ... trong thư mục `src/routes`



Step 3: Defining routes

```
// main.jsx

import { createBrowserRouter } from "react-router-dom";
import Home from "./routes/Home";

const router = createBrowserRouter([
    { path: "/", element: <Home />, },
    // ... Khai báo thêm các routes
]);
```

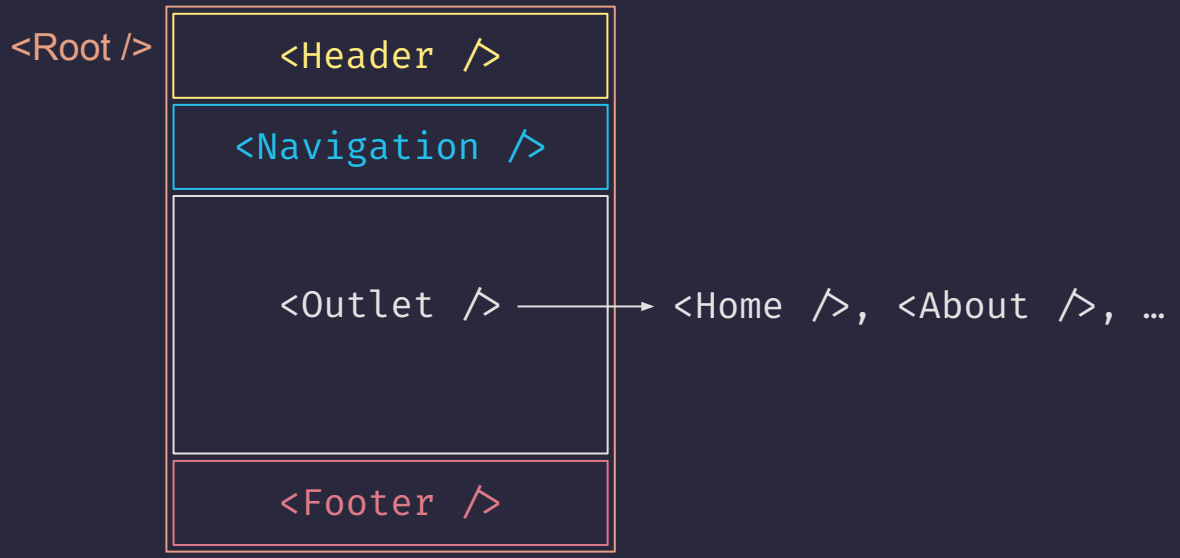
Step 4: Adding router

```jsx
// main.jsx

// Import thêm Provider
import { RouterProvider, createBrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")).render(
    <React.StrictMode>
        <RouterProvider router={router}></RouterProvider>
    </React.StrictMode>
);
```

Step 5: Root route

- Các trang thường chia sẻ chung bố cục và một số thành phần như `<Header />`, `<Navigation />`, `<Footer />`, ...
- Khai báo `<Root />` là route cấp cao nhất, chứa bố cục và các thành phần chung giữa các page
- Các pages được khai báo trong thuộc tính `children` của `<Root />` (nested routes)
- Sử dụng component `<Outlet />` trong `<Root />` giống như placeholder cho biết vị trí mà các page con sẽ được hiển thị
- Chỉ định component mặc định được hiển thị (khớp với URL của `<Root />`) với thuộc tính `index`

Step 5: Root route



Step 5: Root route

```jsx
// src/routes/Root.jsx
import { Outlet } from "react-router-dom";

export default function Root() {
    return (
        <div className="root">
            <Header /> {/* Các thành phần chung*/}

            <Outlet />

            <Footer /> {/* Các thành phần chung */}
        </div>
    );
}
```

```jsx
// main.jsx
const router = createBrowserRouter([
    {
        path: "/",
        element: <Root />,
        children: [
            { index: true, element: <Home /> },
            { path: "about", element: <About /> },
        ],
    },
]);
```

Nested route có thể sử dụng relative path giúp đường dẫn ngắn gọn hơn

Step 6: Routing

- Sử dụng component `<Link />` hoặc `<NavLink />` để tạo liên kết điều hướng
- Đường dẫn (path) chỉ định bởi props `to`
- `<Link />` và `<NavLink />` gần như tương tự nhau, điều khác biệt là `<NavLink />` tự động thêm `class="active"` nếu đường dẫn hiện tại khớp với đường dẫn trong `to`
- `<NavLink />` thường sử dụng cho các liên kết trong menu điều hướng chính
- `<Link />` thường sử dụng cho các liên kết cho card sản phẩm, card phim, ... không cần phải highlight theo URL

Step 6: Routing

```jsx
// src/components/Header.jsx
import { NavLink } from "react-router-dom";

export default function Header() {
    return (
        <header className="header">
            <div className="logo">Logo</div>

            <nav className="nav">
                <NavLink className='nav-link' to="/">Home</NavLink>
                <NavLink className='nav-link' to="/about">About</NavLink>
                {/* Thêm các liên kết điều hướng đến các trang khác */}
            </nav>
        </header>
    );
}
```

Step 6: Dynamic route

- Dynamic route cho phép khai báo một URL chung để khớp bất kỳ URL có mẫu giống nhau
- Thường dùng cho các trang như chi tiết bài viết, sản phẩm, tập phim, ... chỉ thay đổi một phần (segment) trong URL, ví dụ `/posts/1`, `/posts/2`, ...
- URL params là các phần thay đổi trong URL, được khai báo dạng `:paramName`, ví dụ `/posts/:id`
- Một URL có thể có nhiều params khác nhau (mỗi param tương ứng 1 phần - segment - trong URL)
- Param thường dùng để tải dữ liệu, lọc dữ liệu, ...

Step 6: Dynamic route

```jsx
// main.jsx
const router = createBrowserRouter([
    {
        path: "/",
        element: <Root />,
        children: [
            // other routes
            { path: "posts", element: <PostList /> },
            { path: "posts/:postId", element: <PostDetail /> },
            { path: "posts/:postId/authors/:authorId", element: <Profile /> },
        ],
    },
]);
```

Step 6: Dynamic route

```jsx
// src/routes/PostList.jsx
import { Link } from "react-router-dom";

export default function PostList() {
    return (
        <div className="post-list">
            {posts.map((post) => (
              <Link to={`/posts/${post.id}`}>
                {/* Show post info */}
              </Link>
            ))}
        </div>
    );
}
```

```jsx
// src/routes/PostDetail.jsx
import { useParams } from 'react-router'

export default function PostDetail() {
  const params = useParams(); // { id }

  // Sử dụng id để load dữ liệu

  return (
    <div>Hiển thị thông tin bài viết</div>
  )
}
```

Step 7: Handling not found

```jsx
// main.jsx
const router = createBrowserRouter([
    {
        path: "/",
        element: <Root />,
        children: [
            // other routes
            { path: "*", element: <PageNotFound /> },
        ],
    },
]);
```

Step 8: Context

- Cho phép chia sẻ trạng thái chung giữa các trang
- Được tích hợp sẵn vào Outlet
- Có thể sử dụng các cách thức quản lý trạng thái khác để chia sẻ dữ liệu giữa các trang, ví dụ Redux, Recoil, ...

Step 8: Context

```jsx
// src/routes/Root.jsx
export default function Root() {
  const [state, setState] = useState({});

    return (
        <div className="root">
          <Header /> {/* Các thành phần chung*/}

          <Outlet context={state} />

          <Footer /> {/* Các thành phần chung */}
        </div>
    );
}
```

```jsx
// src/routes/Home.jsx
import { useOutletContext } from 'react-router'

export default function Home() {
  const state = useOutletContext();

  return (
    <div>Home</div>
  )
}
```

03

HOOKS

Giới thiệu một số hooks thường dùng

useSearchParams()

- Quản lý và đọc dữ liệu từ query param (?)
- Lưu ý vấn đề phổ biến khi làm việc với query param là đồng bộ URL với form state, các ví dụ điển hình là:
  - Khi bấm nút back, mặc dù URL thay đổi nhưng trạng thái form vẫn giữ nguyên
  - Khi reload trang, mặc dù URL giữ nguyên (có query params) nhưng trạng thái form thì bị mất
- Với các giá trị dạng mảng (ví dụ checkbox), sử dụng createSearchParams để tạo query params

useSearchParams()

```jsx
// src/routes/PostList.jsx
import { useSearchParams } from "react-router-dom";

export default function PostList() {
    const [searchParams, setSearchParams] = useSearchParams();

    // URL: https://.../posts?title=react
    const title = searchParams.get("title");

    useEffect(() => {
        // do something with title
    }, [title]);

    // return JSX
}
```

useSearchParams()

```
const handleChange = (e) => {
    const value = e.target.value;

    if (value.trim().length == 0) {
        searchParams.delete(e.target.name);
    } else {
        searchParams.set(e.target.name, value);
    }

    setSearchParams(searchParams);
};
```

Đồng bộ URL và trạng thái của input

```
<input
    type="search"
    placeholder="Search post"
    name="title"
    value={title ?? ""}
    onChange={handleChange}
/>
```

useNavigate()

- Sử dụng để chuyển hướng trang trong các hàm xử lý sự kiện, hay trong `useEffect()`, ví dụ sau khi submit form thành công, khi không có thông tin user, ...
- `useNavigate()` trả về hàm `navigate` sử dụng để chuyển hướng đến một URL khác
- Hàm `navigate` chỉ nên được gọi trong `useEffect()` hay trong các hàm xử lý sự kiện
- Tham số cho `navigate` có thể là một số (trong history stack) hoạt động tương tự như bấm nút back trên trình duyệt

useNavigate()

```jsx
// src/routes/Login.jsx
export default function Login() {
    const navigate = useNavigate();

    const handleSubmit = (e) => {
        e.preventDefault();
        login().then(() => navigate("/"));
    };

    return <form onSubmit={handleSubmit}></form>;
}
```