React 4 – Client Side Routing and React Router (Detailed Notes with Examples)

---

1. Intuition for Frontend Routing

Frontend routing means deciding **which UI component to show based on the URL**, without requesting a new page from the server. In React SPAs, the browser loads the app once and React controls navigation. When the URL changes, React Router swaps components instead of reloading the page. This makes navigation fast and smooth. The server is no longer responsible for page navigation. React Router handles everything on the client side.

---

2. Request–Response Cycle for a React App

When a user enters a website URL, the browser sends a request to the server. The server responds with an HTML file and a React JavaScript bundle. React then takes control of rendering the UI. After this, page navigation does not hit the server again. Only API calls are made for data. This is the foundation of a Single Page Application.

---

3. Behaviour of SPA (Single Page Application)

In an SPA, the page loads once and never refreshes fully. Navigation updates only part of the UI. Application state like login or cart is preserved. URL changes are handled internally. This improves speed and user experience. React Router enables this behavior in React apps.

---

4. UI vs Data Concept

In React apps, UI and data are treated separately. UI consists of components, layouts, and styles. Data comes from APIs or backend services. UI is usually loaded once during startup. Data is fetched whenever required. This separation makes apps efficient and scalable.

---

5. Initial Bundle and Loaders

On first load, React downloads the core bundle. Some components may not be visible immediately. Loaders or placeholders are shown while data loads. This prevents blank screens. Once data arrives, UI updates automatically. This gives users a smooth experience.

---

6. Installing React Router DOM

React Router DOM is required for client-side routing in web apps.

Installation

```
npm i react-router-dom
```

This library provides `BrowserRouter`, `Routes`, `Route`, `Link`, `useParams`, and `Navigate`.

---

7. BrowserRouter Setup (Very Important)

`BrowserRouter` enables routing using browser history. It must wrap the entire application. Without it, routes will not work.

main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { BrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

---

8. Creating Basic Routes

Routes map URLs to components. Each route renders a component when its path matches the URL.

Example Components

```
function Home() {
  return <h2>Home Page</h2>;
}

function About() {
  return <h2>About Page</h2>;
}

function Listing() {
  return <h2>Listing Page</h2>;
}
```

Routing File (Routing.jsx)

```
import { Routes, Route } from "react-router-dom";

function Routing() {
  return (
    <Routes>
      <Route path="/" element={<Home />} />
      <Route path="/about" element={<About />} />
      <Route path="/listing" element={<Listing />} />
    </Routes>
  );
}

export default Routing;
```

---

9. Wildcard Route (404 Page)

Wildcard routes catch invalid URLs. They prevent blank screens.

```
function PageNotFound() {
  return <h2>404 Page Not Found</h2>;
}

<Route path="*" element={<PageNotFound />} />
```

React Router always tries exact routes first, then wildcard.

---

10. Navigation using Link

`Link` allows navigation without page reload.

Navbar Example

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

<nav>
  <Link to="/">Home</Link>
  <Link to="/about">About</Link>
  <Link to="/listing">Listing</Link>
</nav>
```

This updates the URL and UI instantly.

---

11. Why Link Instead of <a>

`<a>` reloads the page and resets state. `Link` keeps SPA behavior intact. `Link` avoids unnecessary network requests. It integrates with React Router automatically. Always use `Link` for internal navigation.

---

12. Dynamic / Template Routes

Dynamic routes allow variable URLs.

Example

```
<Route path="/users/:id" element={<User />} />
```

This single route handles `/users/1`, `/users/2`, etc.

---

13. useParams Hook

`useParams()` extracts route parameters.

Example

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

function User() {
  const params = useParams();
  return <h2>User ID: {params.id}</h2>;
}
```

---

14. Fetching API Data with Dynamic Routes

Dynamic routes are commonly used with APIs.

Example (Fake Store API)

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";

function User() {
  const { id } = useParams();
  const [user, setUser] = useState(null);
```

```
  useEffect(() => {
    fetch(`https://fakestoreapi.com/users/${id}`)
      .then(res => res.json())
      .then(data => setUser(data));
  }, [id]);

  if (!user) return <h3>Loading...</h3>;

  return (
    <>
      <h3>{user.username}</h3>
      <p>{user.name.firstname} {user.name.lastname}</p>
    </>
  );
}
```

---

15. Redirecting Routes using Navigate

Redirects move users automatically.

Example

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

<Route path="/old-path" element={<Navigate to="/" />} />
```

No reload occurs. Navigation remains client-side.

---

16. Folder Structure (Recommended)

```
src/
 ├─ components/
 │    ├─ Home.jsx
 │    ├─ About.jsx
 │    ├─ Listing.jsx
 │    ├─ User.jsx
 ├─ Routing.jsx
 ├─ App.jsx
 └─ main.jsx
```

This structure improves readability and scalability.

---

17. Practice Flow (How You Should Practice)

1. Create pages (Home, About, Listing)
2. Setup BrowserRouter
3. Add Routes
4. Add Link-based navigation
5. Add dynamic route
6. Fetch API data using params
7. Add wildcard route
8. Add redirect

If you follow this order, routing will become natural to you.

Final Note

Practice by changing URLs manually. Break routes intentionally and fix them. Add console logs to understand flow. Routing is not memorization—it is understanding URL to UI mapping. Once this is clear, advanced topics become easy.